Artificial Intelligence Methods in the Environmental Sciences

Sue Ellen Haupt
Antonello Pasini
Caren Marzban
Editors

Springer

Artificial Intelligence Methods in the Environmental Sciences

Sue Ellen Haupt · Antonello Pasini · Caren Marzban

Editors

Artificial Intelligence Methods in the Environmental Sciences

Springer

Editors
Sue Ellen Haupt
Pennsylvania State University
Applied Research Laboratory
Box 30
State College, PA 16804-0030
USA

Antonello Pasini
National Research Council
Institute of Atmospheric Pollution
Via Salaria Km. 29.300
00016 Monterotondo Stazione
Rome
Italy

Caren Marzban
University of Washington
Dept. of Statistics
Box 354322
and the Applied Physics Laboratory
Seattle, WA 98195-4322
USA

ISBN 978-1-4020-9117-9 (HB) ISBN 978-1-4020-9118-6 (PB)
e-ISBN 978-1-4020-9119-3

Library of Congress Control Number: 2008935886

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Preface

This book is the product of the efforts of a group of people affiliated with the Committee on Artificial Intelligence Applications to Environmental Science of the American Meteorological Society (AMS). This committee has sponsored four two-day short courses or workshops on the applications of Artificial Intelligence (AI) techniques in the environmental sciences and is currently planning its seventh conference. Although it began as a loose knit group of AI practitioners, it has evolved with the field to help guide the development of new methods and their applications in the environmental sciences. Several authors and editors have authored or coauthored full books of their own, some on the AI methods themselves and others related to specific environmental science topics.

The idea to band together to write a book grew during the planning phase for the third Short Course held in Atlanta, GA in January 2006 in conjunction with the Annual Meeting of the AMS. While preparing materials for a short course, we wondered why not also compile our lectures and demonstrations in book form? The idea began there, but continued to expand. We did not want to be merely an amalgamation of different topics, but rather to have a sensible organization and a common theme. The organization was suggested by the Springer reviewers who convinced us to divide the book into a teaching portion (Part I, which motivates and introduces the primary AI techniques) and a showing portion (Part II, which comprises example applications of the techniques). Developing the common theme required many email exchanges between the editors and a final real-life discussion between authors at the 2006 Short Course. Thus was born the "red thread" that weaves the book together. The red thread blends cultures in several senses. The American editors talked about weaving a thread through the book to tie it together while the Italian editor wrote about drawing a "red line" through the book to make the connections clear. So the "red thread" is a cultural blending of ideas in very much the same sense as AI is a blending of traditional techniques with the more recent ones for making sense of data. We wish to be careful to avoid too much of a contrast, since in some sense, AI is just a new terminology for codifying very old ways of making sense of data. Chapter 1 emphasizes this point by showing the development of traditional dynamics-based modeling for weather forecasting, then developing the basic data-based AI concepts. In a similar vein, Chapter 2 relates some traditional statistical models with more recent ideas developed in AI and machine learning circles.

Although our styles of presentation may differ, this book was edited to maintain a consistent "spirit." We have used the red thread to weave the fabric of the methods

into a tapestry that pictures the "natural" data-driven AI methods in the light of the more traditional modeling techniques.

The editors wish to thank the Springer editors; the American Meteorological Society, particularly the Scientific and Technological Activities Commission that sponsors the Committee and has supported our conferences, short courses, and workshops; and particularly, all the participants in our events who have helped us to continue to think of fresh ideas for applications.

Sue Ellen Haupt, Antonello Pasini, and Caren Marzban
2008

Contents

Part I
Introduction to AI for Environmental Science

Environmental Science Models and Artificial Intelligence

1

Sue Ellen Haupt, Valliappa Lakshmanan, Caren Marzban, Antonello Pasini, and John K. Williams

1.1 On the Nature of Environmental Science

Environmental science is one of the oldest scientific endeavors. Since the dawn of humanity, along with the ability to reason came the ability to observe and interpret the physical world. It is only natural that people would observe patterns then build basic mental models to predict a future state. For instance, records indicate that there has long been a version of the adage "Red at night, sailor's delight. Red in the morning, sailors take warning."[1] This saying is a simple predictive model based on generations of experience, and it often works. Over time people noted relationships between observations of the sky and subsequent conditions, formed this mental model, and used it to predict future behavior of the weather (Fig. 1.1).

The age of enlightenment following the Renaissance brought a more modern approach to science. Careful experimentation and observation led to uncovering the physics underlying natural phenomena. For instance, a modern understanding of "Red at night, sailor's delight" is based on the theory of Mie scattering. Light rays are scattered by large dry dust particles to the west in the setting sun. According to this theory, large particles tend to scatter the longer wavelength red light forward more than they do the other frequencies of visible light. The long trajectory of the solar beams through the atmosphere when the sun is at a very low zenith angle (such as at sunset or sunrise) compounds this effect. Thus, when light rays from the setting sun are scattered by large dry dust particles associated with a high pressure system to the west, more red light reaches the observer, and the sky appears red. Since prevailing winds in the mid latitudes (where this adage is common) blow from west to east, more Mie scattering at dusk implies that a dry weather pattern is approaching. "Red in the morning, sailors take warning," refers to a similar process at

Sue Ellen Haupt (✉)
Applied Research Laboratory and Meteorology Department, The Pennsylvania State University, P.O. Box 30, State College, PA 16802, USA
Phone: 814/863-7135; fax: 814/865-3287; email: haupts2@asme.org

Valliappa Lakshmanan
Cooperative Institute of Mesoscale Meteorological Studies (CIMMS), University of Oklahoma and National Severe Storms Laboratory (NSSL), 120 David L. Boren Blvd., Norman, OK 73072, USA
Phone: 405-325-6569; email: lakshman@ou.edu

Caren Marzban
Applied Physics Laboratory and the Department of Statistics, University of Washington, Seattle, WA 98195-4323, USA
Phone: 206-221-4361; fax: 206-685-7419; email: marzban@stat.washington.edu

Antonello Pasini
CNR – Institute of Atmospheric Pollution, Via Salaria Km 29.300, I-00016 Monterotondo Stazione (Rome), Italy
Phone: + 39 06 90672274; fax: + 39 06 90672660;
email: pasini@iia.cnr.it

John K. Williams
Research Applications Laboratory, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307, USA
Phone: 303-497-2822; fax: 303-497-8401;
email: jkwillia@ucar.edu

[1] For instance, see the quote in Matthew 16:1-2 of the Bible: "He replied, 'When evening comes, you say, 'It will be fair weather, for the sky is red,' and in the morning, 'Today it will be stormy, for the sky is red and overcast' " (NIV).

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

dawn when the low zenith angle in the east would produce more scattering associated with a high pressure system that has already passed, thus suggesting the possibility that a low pressure system is now approaching and wet weather may follow. This example exemplifies the types of physical explanations of observed phenomena that developed in the environmental sciences.

Emergence of modern mathematical techniques gave scientists a new language to describe the natural world. The new physical understanding was codified into partial differential equations (PDEs) that represent the details of the physics. These PDEs can be used to predict the future evolution of a system given its initial state. Modern meteorology began with the development of the primitive equations that describe the conservation of mass, momentum, and energy in the atmosphere. It was necessary to specify the external forcings, including solar insolation and the Earth's rotation. Appropriate boundary and initial conditions had to be applied. Numerical techniques were developed to discretize interlinking equations to form algebraic equations that can be solved simultaneously. In the 1920s, a pioneer of modern meteorology, L.F Richardson, attempted to integrate the full primitive equations by hand. Unfortunately, he did not know about the importance of filtering the equations to avoid the effects of fast gravity and acoustic waves, which caused his integration to "blow up" (Richardson 1922). In spite of the fact that he obtained a very unphysical solution and that the hand integration of the equations took much longer than the weather itself, Richardson made a huge impact on the science of weather forecasting through demonstrating that the equations could be used for prediction and by foreseeing the impact of modern parallel computing. In his 1922 treatise, he imagines an orchestration of human "computers" for numerical weather prediction:

Fig. 1.1(a) "Red at night, sailor's delight." Source: Copyright B.A. Haupt (2005)

Fig. 1.1(b) "Red in the morning, sailor's take warning." Source: Copyright B.A. Haupt (2005)

Imagine a large hall like a theatre, except that the circles and galleries go right round through the space usually occupied by the stage. The walls of this chamber are painted to form a map of the globe. The ceiling represents the north polar regions, England is in the gallery, the tropics in the upper circle, Australia on the dress circle and the antarctic in the pit. A myriad of computers are at work upon the weather of the part of the map where each sits, but each computer attends only to one equation or part of an equation. The work of each region is coordinated by an official of higher rank. Numerous little "night signs" display the instantaneous values so that neighbouring computers can read them. Each number is thus displayed in three adjacent zones so as to maintain communication to the North and South on the map. From the floor of the pit a tall pillar rises to half the height of the hall. It carries a large pulpit on its top. In this sits the man in charge of the whole theatre; he is surrounded by several assistants and messengers. One of his duties is to maintain a uniform speed of progress in all parts of the globe. In this respect he is like the conductor of an orchestra in which the instruments are slide-rules and calculating machines. But instead of waving a baton he turns a beam of rosy light upon any region that is running ahead of the rest, and a beam of blue light upon those who are behindhand.

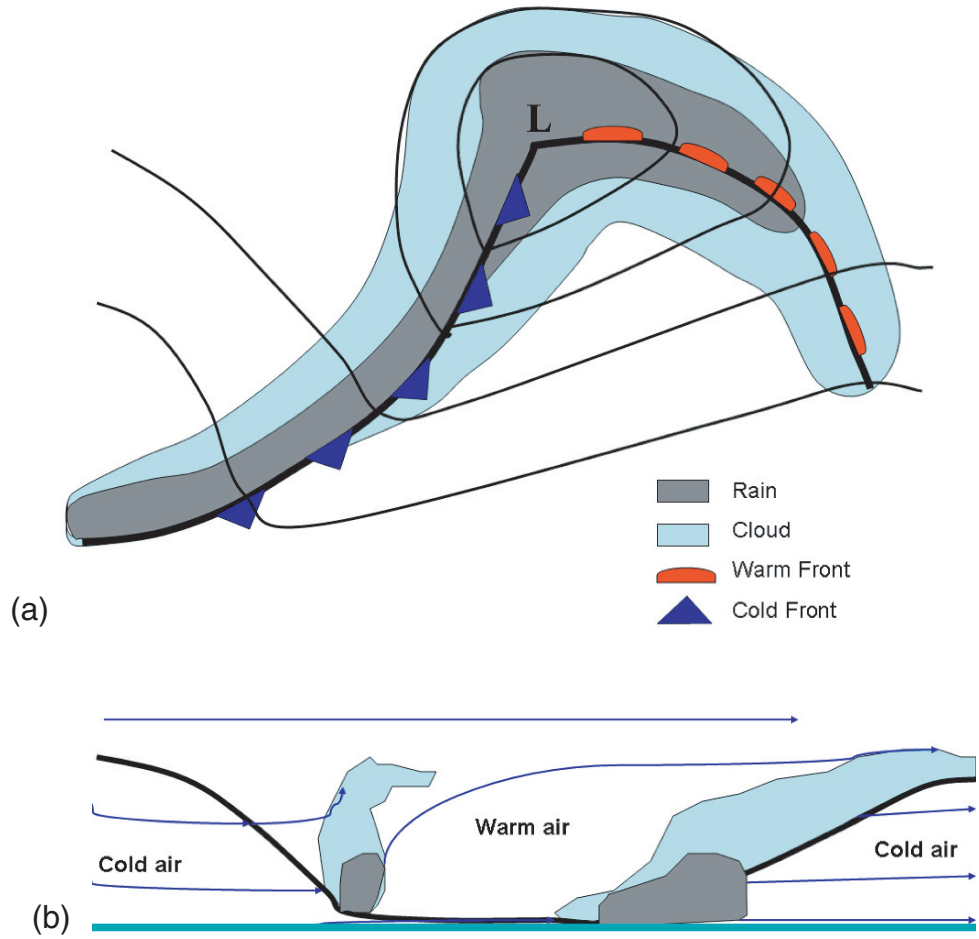


Fig. 1.2 Schematic of the Norwegian Cyclone Model. (a) Horizontal and (b) vertical cross-sections

> Four senior clerks in the central pulpit are collecting the future weather as fast as it is being computed, and despatching it by pneumatic carrier to a quiet room. There it will be coded and telephoned to the radio transmitting station. Messengers carry piles of used computing forms down to a storehouse in the cellar. (Richardson 1922)

At the same time, weather observations continued and more formal models of repeated patterns were formulated. For instance, in meteorology, the Norwegian cyclone model emerged in the early 1920s, attributed primarily to V. Bjerknes and J. Bjerknes (Reed 1977). It describes a weather system in terms of warm fronts, cold fronts, and occluded fronts (see Fig. 1.2). In between the warm and cold fronts is a warm sector. High cloudiness is expected to precede the warm front, which brings in warm southerly winds sometimes preceded by a band of light showers. The warm air rises over both the old cold air being pushed out as well as the new cold air ushered in by the cold front. This conveyer belt of rising warm air produces convection, cooling, condensation, and rain. Convective motion causes showers in the warm sector and deep convection near the cold front often results in violent thunderstorms. So before scientists could accurately model cloud physical processes and their relationship to precipitation mathematically, they could use the Norwegian cyclone model to interpret the skies and predict associated precipitation patterns. Once again, lots of observations came together to form a more formal model of atmospheric phenomena useful for prediction.

With the advent of electronic computers in the 1950s, meteorological research returned to numerical weather prediction, this time aided by the rapid calculations of a machine. The first operational computer, the ENIAC (Electronic Numerical Integrator

and Computer) at Aberdeen Proving Grounds became the forecasting tool of Jules Charney, John von Neumann, and R. Fjortoft. They wrote the dynamical relationships in the form of the barotropic vorticity equation, which does not include the fast gravity and acoustic waves that had plagued Richardson. Their efforts met success in 1950 when they produced the first useful numerical weather forecast on the ENIAC (Charney et al. 1950; Lorenz 2006). The field of Numerical Weather Prediction (NWP) was thus born. The field advanced (and continues to advance) through the development of finer resolution models with smaller time steps that can capture more details of the physics. These NWP models have grown to include cloud microphysics, details of radiative transfer, interactions with the biosphere and cryosphere, and dynamic oceanic forcing, among other important dynamical and physical processes. For quite some time, the accuracy of short term prediction continued to improve. Some even thought that if researchers could continue to better define the physics and refine the spatial and time scales, they would eventually be able to perfectly predict the weather arbitrarily far in advance.

This hope was dashed by the famous rediscovery by Ed Lorenz at MIT of what has become known as chaos theory (Lorenz 1963). Dr. Lorenz was studying a version of the atmospheric equations simplified to just three nonlinearly coupled ordinary differential equations (Gleick 1987). He started a numerical integration of his "simple" system of equations then went to lunch. When he came back, his computer run had stopped and he wanted to restart it. He went back several steps and entered the solution at that time as an initial condition for the next run. Then he realized that the numbers coming out of the computer did not quite exactly match those from the succeeding steps of the previous run. Computers were supposed to exactly reproduce their prior computations. What was different? The cause of the disparity was eventually traced to the fact that when he typed in the numbers to restart the program, he did not type in all the digits. The small round-off error grew with time. What he discovered was that in nonlinear dissipative systems, there is sensitivity to initial conditions. This sensitivity means that when a computer model is started with two sets of initial conditions that differ only slightly, the resulting solutions can diverge rapidly in time. Lorenz's observation led to the discovery of chaotic flow and of the limits to predictability in systems of equations such as those representing the atmosphere. What does this imply for forecasting the weather by integrating the dynamical equations? It implies that there is a limit to predictability. No matter how big and fast our computers become, we cannot expect to ever forecast accurately beyond this inherent limit. There is hope, however, that we can find the "strange attractor" of the system, and we are assured that the solution will be somewhere on that manifold of possible solutions. Thus, it might be possible to say something about the likely weather pattern, even without pinpointing the specific conditions. Figure 1.3 pictures the "butterfly"-shaped strange attractor of Lorenz's three equation system in the chaotic regime.

Several new research fields emerged to deal with the discovery of the limits to predictability. The first involves trying to transform atmospheric measurements into an appropriate "initial condition" for a forecast model run. This process is known as initialization. Various complex statistical techniques can be used to first assimilate and interpolate the monitored data to the grid of the model and find a best fit to the steady state model equations. In this manner, the initial condition is tuned to have the lowest possible error to help keep it closer to the solution that could be represented by the model. (See Daley 1991 or Kalnay 2005 for more discussion.)

A second new direction that has emerged is using statistical and data-based approaches to weather prediction. One class of methods is model output statistics (MOS), which compares past model predictions to corresponding records of the observed actual conditions to tune the model or adjust its forecasts (Glahn and Lowry 1972). Another class of methods involves addressing sensitivity to initial conditions head-on by using multiple initial conditions to initialize ensembles of NWP model runs. The forecaster must then interpret the multiple forecasts to make a prediction, but is provided a measure of uncertainty via their variance. At the lowest level, this approach has returned to using the human art of forecasting to distinguish between and synthesize the ensemble of solutions. Of course, it was not long until advanced statistical techniques attempted to emulate that human process by using methods reminiscent of model output statistics to tune the ensemble model runs (Gnieting and Raftery 2005).

Now that statistical techniques have been applied to the forecasting problem, it is natural to ask whether

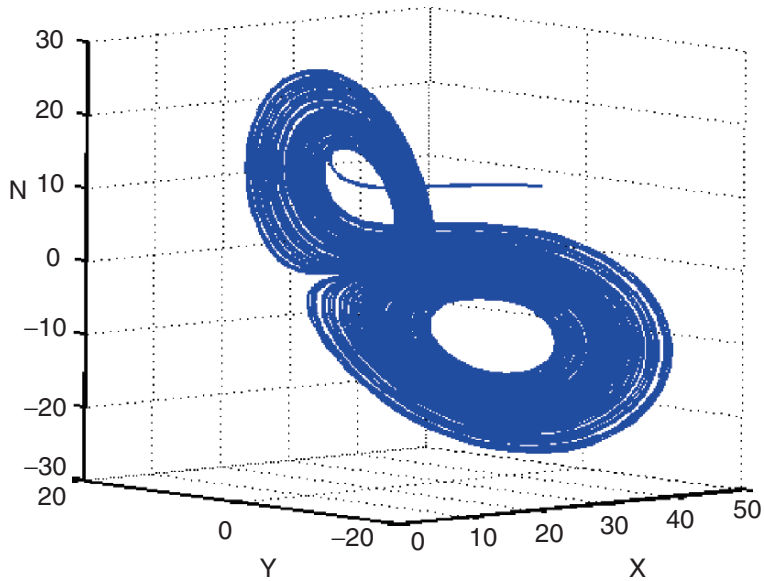


Fig. 1.3 The Lorenz attractor

a completely statistical approach could be developed. Some recent research has attempted to do that to a limited extent. Among the nascent techniques are empirical methods that are no longer based on the deterministic physics, but instead seek ways to glean past information for predictive modeling and to code it into PDEs. Such empirical stochastic methods have been successful at predicting some major atmospheric phenomena such as the El Nino/Southern Oscillation (Penland and Matrosova 1998) and for determining how the atmosphere responds to an imposed forcing (Branstator and Haupt 1998).

These developments leave environmental science in general, and numerical weather prediction in particular, at a crossroads. There are two totally different approaches to forecasting: the first is deterministic or dynamics-based and depends on our ability to write all the dynamical and physical processes mathematically and to discretize them so that they can be solved numerically; the second is empirical or data-based, and depends on the available data and how we choose to use it statistically for prediction. Both approaches typically require large scale computing resources.

Even newer are forecasting techniques that attempt to return to the early methods of recognizing useful patterns without employing formal statistical theory. These methods are empirical data-based methods that do not directly depend on dynamics, but instead seek to model some natural process. The goal of this book is to demonstrate how these "artificial intelligence" methods can be successful at recognizing patterns, performing regression for the purpose of prediction, and optimizing solutions to difficult nonlinear problems. These methods are different from the prior directions of numerically integrating carefully derived dynamical physics equations. Researchers who use artificial intelligence methods are never able to include every tiny detail of the physics in their equations. There is inherent uncertainty in both the measurements and the equations. But looking at how the environment has previously changed will help us predict how it will change the next time, even without modeling all the details. We can use the patterns we recognize, whether they are cold fronts and warm fronts or clustering in some phase space to tell us what typically happens in such a situation. Lorenz's work (1963) implies that we will never be able to exactly predict the future behavior of these complex systems, but it does not inform us whether our dynamics-based methods are likely to be any better than our data-based methods. Many researchers have been highly trained in the physical dynamics-based deterministic methods, and that is certainly helpful background for our adventure into the empirical data-based AI methods.

These developments in weather forecasting are indicative of the types of challenges that arise in the broader field of environmental science. Solid earth geophysicists have explored physical models,

data-based models, and artificial intelligence methods to advance their field. So have wildlife biologists, ecologists, water resource managers, air pollution experts, and space scientists, among the many other branches of environmental science.

Many environmental fields deal with complex systems: the atmosphere and the climate system, for instance, or, even more complex living ecosystems and their environment. From a dynamical point of view, we can look at these systems as built on a very high number of component subsystems that interact with each other with feedback. The dynamical approach to grasping this complexity is based on a "decomposition-recomposition" strategy: look at the fundamental subsystems and interactions, phenomena and processes; describe them via dynamical equations; and, finally, simulate the complex system in a computer by means of one or more systems of coupled equations (see Pasini 2005 for an introduction to this dynamics-based approach to modeling in weather and climate studies).

Due to the complexity of the physical systems analyzed, dynamics-based models can suffer from significant problems, such as the difficulty of correctly balancing the "strength" of the many interactions and feedbacks. Often, the modeler must fine-tune the values of the coupling parameters to obtain realistic behavior: obviously, this makes it impossible to uniquely reconstruct the simulated system, thus weakening the reliability of the simulation results coming from the deterministic model. Therefore, the challenge of complexity seems to hint at the need for a more phenomenological approach where the system can be considered as a whole and its behavior can be analyzed in terms of the evolution in time of some representative or important variables of the system, subject to all the interactions and feedbacks. In this framework, artificial intelligence methods have proven useful.

1.2 On the Nature of Artificial Intelligence

Artificial intelligence (AI) has grown out of modern computing combined with a plethora of data to interpret and engineering problems to solve. In some sense, it returns to the earlier methods of analyzing data and trying to build predictive models based on empirical data in a "natural" way. In this sense, AI techniques are typically more data-based than dynamics-based. This use of data can make fast, robust, and skillful forecasts possible in domains that might be intractable by a dynamics-based approach. As Einstein once remarked, "So far as the laws of mathematics refer to reality, they are not certain. And so far as they are certain, they do not refer to reality" (Einstein 1922).

A 19th century scientist would be astonished by the capability of today's computers – solving multivariable PDEs, running numerical models of the atmosphere, or simulating debris flows, for instance. Yet, that scientist would be equally astonished by the incapabilities of modern computers – their limitations in recognizing a face in a crowd or responding to spoken language, for example. Humans, on the other hand, find that recognizing faces is a cinch but solving multi-variable equations is hard. This is not how it was supposed to be. The earliest computers were billed as machines that could *think*. Indeed, AI has been defined as enabling machines to perceive, understand, and react to their environments. Although perceptive humanoid robots have been a staple of science fiction for many decades, they are still quite impractical. Rather, successful applications of AI have concentrated on single tasks, such as optimally managing the gates at an airline terminal or successfully classifying cells as benign or cancerous.

AI started out by attempting to build upon Aristotelian ideas of logic. Thus, initial research emphasized induction and semantic queries. The goal was to build a system of logic by which computers could "reason" their way from simple bits of data to complex conclusions. After all, humans do this sort of reasoning effortlessly. However, it slowly became apparent that there is more to human reasoning than just induction. Humans, it turns out, are naturally good at many things in ways that current computer designs may never match.

AI researchers scaled back their overly ambitious initial goal to that of building systems that would complement human users and do tasks that humans found onerous. Computers are good at unflaggingly processing reams of data and performing complex computations, but poor at obtaining a holistic view of systems. Humans, good at higher-level thinking, find it hard to do mind-numbing calculations. AI researchers also approached the problem with a new awareness of the impressive performance of biological systems. Thus, many of the new AI approaches were intentionally

modeled on the way human experts thought or behaved or on how underlying biological systems such as the human brain worked.

Rather than building an AI system that would replace a human's multifaceted capabilities, researchers concentrated on building special purpose systems that could do one thing well. Thus, instead of building a system that would conduct a wellness check on a patient, for example, they developed a system that would determine an appropriate drug dosage for a cancer patient. The solutions to such targeted problems were called expert systems, because they encoded the rules that an expert in the field would follow to come to his or her conclusions. Computers proved capable of quickly and objectively determining answers to problems where the methodology was precisely defined. The rules in such expert systems are often in the form of decision trees, where the answer to one question narrows down the possibilities and determines what question is asked next, until all possible conclusions but one (or a few) are eliminated.

One fundamental problem in expert systems is how to represent and apply the domain knowledge of experts. Experts often state their knowledge verbally using imprecise words like "not very hot" and "less water." Such words do not lend themselves well to decision trees since the ambiguity can span multiple branches of the tree at each step. This issue is addressed by another AI technique: fuzzy logic. Fuzzy logic provides a framework for encoding imprecise verbal rules – such as those provided by subject domain experts – and aggregating them to yield a final answer. It also allows partial, ambiguous or uncertain evidence to be maintained and efficiently synthesized in the production of the final result. The most celebrated successes of fuzzy logic have been in Japan, where numerous engineering control systems have been built based on encoding expert knowledge in fuzzy rules that are then combined for prediction or control.

Fuzzy logic can be used to create automated decision support systems that model what a human expert would do under similar circumstances. Because humans are considerably skilled at recognizing patterns and often understand the underlying processes that lead to the data, the verbal rules formulated by human experts are often quite robust, even to unseen data and unanticipated situations. A fuzzy logic system, by piggy-backing on such effective analysis, can possess considerable skill. Because fuzzy logic systems are relatively simple to encode and do not require training datasets, they are also fast to create and implement. Another advantage of these systems, often a deciding factor in many applications, is that the fuzzy rules and their synthesis can be naturally interpreted by a human expert. If a training dataset is available, it can be used to optimize the fuzzy logic algorithm's parameters, though this step is not required. Thus, a fuzzy logic system can provide considerable skill and a human-understandable, tunable system for very little investment.

Fuzzy logic systems often provide good solutions to problems for which reliable expert knowledge is readily available and can be represented with verbal rules or heuristics. While there are many situations where this is the case, there are also many others in which either no experts are available or their knowledge cannot easily be represented with verbal rules. One might gauge the suitability of a fuzzy logic approach by assessing whether different experts tend to agree on data cases; if they don't, it might be necessary to code and evaluate multiple fuzzy logic algorithms to represent the range of solution methodologies, which might not be practical. It may also be difficult to determine whether the verbal rules that experts identify is really all that they use to come to their conclusion. Humans often underestimate the role of intuition or the subconscious knowledge brought to bear on a problem. Moreover, many domains in the environmental sciences are exceedingly complex and poorly understood to begin with, so a method capable of automatically recognizing patterns from data may be more appropriate.

Fortunately, another AI method excels at modeling complex, nonlinear systems based on data – the neural network (NN), Like many AI methods, NNs are biologically inspired. The name comes from the fact that they were initially modeled on the way neurons fire, with the accumulated firings of many neurons together determining the brain's response to any particular set of stimuli. The most common architecture used in NNs comprises three layers of neurons – an input layer, a layer of "hidden nodes" and a final output layer. Such an NN can represent any continuous function on a compact domain arbitrarily closely, even a nonlinear one, if it has enough hidden nodes – though choosing the optimal number of hidden nodes for a particular problem may require some effort (Cybenko 1989).

Feed-forward NNs are members of the class of supervised learning machines. In a process called "training", such a learning machine is presented with patterns – sets of inputs and target values, or ideal output corresponding to desired responses to those inputs. The target values may either be provided by an expert in the field, or can be obtained from field surveys, measurements and other information; as such, the target values are often referred to as "ground truth." At the end of training, if all goes well, the learning machine will have created a function that approximately maps the training inputs to the associated targets. Subsequently, when this function is presented with a new set of inputs, it determines a response based on the evidence generalized from the training specimens. Thus, if viewed as an expert system, NNs learn previously unknown relationships or knowledge that experts may not be able to represent with verbal rules. This is because supervised learning machines approximate expert learning behavior, not by approximating the logic that experts use and inexactly describe, but by creating a new mapping to the ground truth associated with the inputs. Specifically, NNs are trained by adjusting their parameters to minimize a cost (or objective) function – a quantity that is usually some function of the difference between the target values and the approximation thereof produced by the network.

Although NNs can represent any continuous function and avoid the problem of depending on expert descriptions by learning data relationships instead, that flexibility comes with a price. First, although the NN learns to approximate the mapping from training samples to target values, the actual function used to represent this approximation is encoded in a large set of connection weights that usually yield no insights. Thus, unlike an expert system, an NN representation is generally not human-understandable, though researchers have found ways to extract approximate rules from an NN in some specific cases (see, for instance, Setiono et al. 2002). The inability to explain in simple terms the behavior of an NN has led to it being called a "black box." However, it is important to point out that this opaqueness is not specific to NNs but applies to many nonlinear models, which may represent the physical world very well but resist being neatly summarized. The fact is that the human desire to explain relationships in simple terms may be inconsistent with the competing requirement to have the most accurate predictions possible, a trade-off that is not peculiar to AI; more details on this will be provided in Chapter 2.

NNs' ability to fit any data places severe requirements on the data necessary for training them. Many NNs have a large number of parameters (weights) that must be estimated during the training phase. The estimates can be highly unreliable if the size of the training data set is not sufficiently large. An abundance of parameters can also lead to overfitting (see Chapter 2), which in turn can adversely affect NNs' performance on "new" data (i.e., not included in the training set). In short, properly training an NN requires lots of data. How much? That question is difficult to answer, but Chapter 2 describes some methods that can help in addressing it.

Another set of biologically-inspired methods are Genetic Algorithms (GAs). They derive their inspiration from combining the concept of genetic recombination with the theory of evolution and survival of the fittest members of a population. Starting from a random set of candidate parameters, the learning process devises better and better approximations to the optimal parameters. The GA is primarily a search and optimization technique. One can, however, pose nearly any practical problem as one of optimization, including many environmental modeling problems. To configure a problem for GA solution requires that the modeler not only choose the representation methodology, but also the cost function that judges the model's soundness. As mentioned above, training an NN usually involves minimizing some cost function, and that process usually requires differentiating the cost function. By contrast, the learning/training process for a GA does not place any restriction on the differentiability of the cost function, so any measure of performance may be used. The GA is also capable of finding optimal solutions to problems such as those in design. Indeed, genetic algorithms may be used to train either an NN or a fuzzy logic system! Using genetic algorithms to train an NN gives us the ability to use non-differentiable cost functions (see Chapter 18), while using GAs to train a fuzzy logic system allows us to improve on human-devised rules by optimizing their parameters.

Another method for inferring the relationship between inputs and targets is to automatically build a decision tree based on the training data set. This approach is also among the fastest in terms of training speed: decision trees can often be trained on

substantial data sets in a fraction of the time required by competing techniques. Decision trees, like fuzzy logic systems, also have the advantage of being human-understandable. Unlike fuzzy logic, however, one doesn't need to know the rules beforehand – the rules are learned from training data. Decision trees fell out of favor because the skill realizable with decision trees often lags what is possible using other supervised learning techniques. Recent advances in machine learning – averaging decision trees trained on subsets of the training sets ("bagging") and continually focusing the training on the training data cases that the decision trees get wrong ("boosting") – have made decision trees viable again, but at the cost that the resulting decision trees are no longer human readable. However, aggregate statistics obtained from decision trees are useful in gaining insights into how the decision trees come to their decisions. This is yet another illustration of the aforementioned trade-off between pure performance and transparency.

One of the problems with all of the above data-based methods is that the data on which they are based are always imperfect, corrupted by measurement noise or other artifacts, as are the "ground truth" answers provided in the training data set. Artificial intelligence and statistical methods are closely related in that they both attempt to extract information from noisy data. AI techniques can be utilized to create a practical representation whereas statistical methods can be used to measure how confident we may be that the extracted representation is correct.

1.3 On the Use of Artificial Intelligence in Environmental Science

It is only natural that AI should find applications in the environmental sciences. Let's return to our historical example of weather forecasting, in particular, precipitation forecasting on timescales of about a day. We described a progression from basic generalizations such as "red at night ..." through modern numerical weather prediction with model output statistics and ensemble runs to help deal with inherent error due to sensitivity to initial conditions. What other methods could be used to predict precipitation as well as to address the many other environmental problems?

In Chapter 17, we will see the application of fuzzy logic to analyzing Doppler radar data and to predicting atmospheric turbulence for aviation users. In Chapter 18, neural networks trained with a genetic algorithm will be demonstrated for building models to predict hail. Chapter 11 describes how the radiation physics model in a climate model can be replaced by a much faster Neural Network model with no degradation in the results. The complexity of the Lorenz strange attractor is modeled with a GA in Chapter 18 and an NN in Chapter 12. Some very specific nonlinear phenomena including the El Nino-Southern Oscillation (ENSO) are modeled by NN-based nonlinear principal components in Chapter 8. The use of NNs for assimilating satellite data is developed in Chapter 9 and a specific application described in Chapter 10. Interpreting climate data using an NN is described in Chapter 12. An NN is also used to model the boundary layer height based on radon data in Chapter 13. Chapter 14 describes how a GA is applied to back-calculate the initial conditions of a toxic release if sensor data are available. Chapter 16 discusses advances in image processing techniques through using AI. Habitat suitability modeling using NNs is described in Chapter 19. These chapters sample the utility of AI techniques in a variety of environmental problems.

This book does not try to cover all of the statistical techniques used for environmental study or prediction (there are already excellent entire books on that topic) but instead concentrates on Artificial Intelligence methods. We describe many of these methods and demonstrate their usefulness on some problems addressed by the authors. But we cannot hope to be exhaustive, for it is a very broad field. Similarly, no attempt is made to cover any particular method in great detail. We instead reference the many good treatises that provide details of the methodologies. In attempting to give an overview of many applications, we are unable to provide depth. What we hope to do is to give the reader an introduction to some AI methods and a sampling of the sorts of things that can be done and invite him or her into the field to help make progress in developing and testing alternative methods for modeling the natural world. Plenty of challenges in environmental science have not yet been addressed, and lots of relatively new AI methods could be applied to meet these challenges. The primary purpose of this book is to describe some of the basic

AI techniques, demonstrate some applications in the environmental sciences, and whet the reader's appetite for trying them on his or her own applications. Let the adventure begin.

References

Baer, F., & Tribbia, J. J. (1977). On complete filtering of gravity modes through nonlinear initialization. *Monthly Weather Review*, *105*, 1536–1539.

Branstator, G., & Haupt, S. E. (1998). An empirical model of barotropic atmospheric dynamics and its response to tropical forcing. *Journal of Climate*, *11*, 2645–2667.

Charney, J. G., Fjortoft, R., & von Neumann, J. (1950). Numerical integration of the barotropic vorticity equation. *Tellus*, *2*, 237–254.

Cybenko, G. V. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, *2*, 303–314.

Daley, R. (1991). *Atmospheric data analysis* (457 pp.). Cambridge: Cambridge University Press.

Einstein, A. (1922). *Sidelights on relativity* (56 pp.). London: Methuen & Co.

Glahn, H. R., & Lowry, D. A. (1972). The use of model output statistics (MOS) in objective weather forecasting. *Journal of Applied Meteorology*, *11*, 1203–1211.

Gleick, J. (1987). *Chaos: Making a new science* (352 pp.). New York: Viking.

Gneiting, T., & Raftery, A. E. (2005). Weather forecasting with ensemble methods. *Science*, *310*(5746), 248–249.

Kalnay, E. (2005). *Atmospheric modeling, data assimilation, and predictability* (341 pp.). Cambridge, UK: Cambridge University Press.

Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, *20*, 130–141.

Lorenz, E. N. (1986). On the existence of a slow manifold. *Journal of the Atmospheric Sciences*, *43*, 1547–1557.

Lorenz, E. N. (1992). The slow manifold. What is it?. *Journal of the Atmospheric Sciences*, *49*, 2449–2451.

Lorenz, E. N. (2006). Reflections on the conception, birth, and childhood of numerical weather prediction. *Annual Review of Earth and Planetery Sciences*, *34*, 37–45.

Lorenz, E. N., & Krishnamurthy, V. (1987). On the nonexistence of a slow manifold. *Journal of the Atmospheric Sciences*, *44*, 2940–2950.

Machenhauer, B. (1977). On the dynamics of gravity wave oscillations in a shallow water model with application to normal mode initialization. *Beitr. Phys. Atmos.*, *50*, 253–271.

Pasini, A. (2005). *From observations to simulations. A conceptual introduction to weather and climate modeling* (201 pp.). Singapore: World Scientific Publishers.

Penland, C., & Matrosova, L. (1998). Prediction of tropical Atlantic sea surface temperatures using linear inverse modeling. *Journal of Climate*, *11*, 483–496.

Reed, R. J. (1977). Bjerknes memorial lecture: The development and status of modern weather prediction. *Bulletin American Meteorological Society*, *8*, 390–399.

Richardson, L. F. (1922). *Weather prediction by numerical process*. Cambridge: Cambridge University Press.

Setiono, R., Leow, W. K., & Zurada, J. M. (2002). Extraction of rules from artificial neural networks for nonlinear regression. *IEEE Transactions on Neural Networks*, *13*, 564–577.

Basic Statistics and Basic AI: Neural Networks

2

Caren Marzban

2.1 Introduction

There is a little book with the title "What do you believe, but cannot prove?" (Brockman 2006). In it, the editors compile the answer to that question given by 50 of the greatest thinkers alive. The editors did not solicit my answer, but if they had it might have been "I believe but cannot prove that Artificial Intelligence (AI) and statistics are mostly the same; and when they are not, the differences are within the natural variations occurring within each field." Fortunately, AI and statistics have already been compared and contrasted thoroughly.[1] As a result, there is a large body of knowledge that can be employed to create a proof for my claim. However, I also believe that the body of knowledge accumulated is actually more interesting and fruitful than the original question itself, i.e., whether AI and statistics are the same. In fact, I would claim that it does not matter at all whether AI and statistics are, or are not, the same. One characterization that is probably reasonable is that the difference between AI techniques and traditional statistics is not in kind, but in degree. In other words, one may argue that most techniques belong to a continuum, but with different techniques having different degrees of "AI-ness" or "statistic-ness." By the way, in this chapter, AI is synonymous with machine learning.

Caren Marzban (✉)
Applied Physics Laboratory and Department of Statistics,
University of Washington,
Seattle, WA 98195-4323, USA
Phone: +(206) 221-4361; fax: +(206) 685-7419,
email: marzban@stat.washington.edu

Certainly, however, any problem dealing with data is inherently statistical. As such, tools and concepts that have been traditionally developed in statistical circles should not be ignored. The concepts of a random variable and a probability distribution (or histogram), or the difference between a population and a sample, are all extremely important in the analysis and modeling of data. For instance, the results of a study that ignores the difference between a sample and a population are apt to be generally unreliable (or ungeneralizable), because the results are based on a single sample taken from a larger population, and therefore do not pertain to the latter. The question of whether one can generalize the results from the sample to the population is again one that statistics is designed to address. Alas, some AI-related studies generally ignore such issues.

The notion of a probability distribution is also important in any data analysis, although some AI practitioners would disagree. Consider the problem of predicting a quantity Y from another quantity X (e.g., wind speed). If Y is a continuous variable, e.g., temperature in Farenheit, then the problem is referred to as a regression problem. By contrast, a classification problem refers to the situation where Y is a categorical quantity, e.g., occurrence or non-occurrence of tornado. The central quantity in both is the conditional probability density $p(Y|X)$, namely the probability of obtaining some value of Y, *given* that X takes some value. Two examples are: the probability of temperature $= 65\,\mathrm{F}$, given that it is cloudy, or the probability of having a tornado, given that wind speed is 10 knots. The whole business of developing a model

[1] Just google "AI Statistics"!

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

for predicting Y is equivalent to the problem of inferring $p(Y|X)$ from sample data. There are a few AI techniques that do not fit this probabilistic paradigm, but then again they do not produce probabilistic predictions anyway.

Why do we care about probabilistic predictions? A probabilistic prediction conveys a measure of uncertainty, which can be taken into account in further decision making. It is important to acknowledge that everything computed from data is subject to uncertainty, and probabilities offer a concise way of conveying that uncertainty. For example, in a search engine we may have a query that matches 10 possible items, but we would like to present the matches in a descending order of importance. Probabilistic predictions readily allow for this type of ranking. Even in so-called distribution-free techniques which produce strictly categorical predictions (e.g., K-Nearest Neighbor; Section 2.7), one often uses some other criterion to allow for the computation of probability.

The main aim of this chapter is to make the reader aware that most data-related tasks are inherently statistical, and so a statistical framework should be adopted. To that end, the focus is on one question that arises in most (if not all) model being tasks, namely "Which model, if any, is best?". The various models could differ in the choice of their predictors, their degree of nonlinearity, or they may be completely different models altogether. Even with a specific measure of goodness in hand (discussed in Chapter 3), that question turns out to be difficult to answer. The partial answers provided here are sufficiently general (and true) to be useful, regardless of which community – statistics or AI – claims their ownership.

Finally, I believe that any sufficiently elementary treatise is indistinguishable from lies.[2] It is possible that I have been excessive in attempting to simplify matters. Therefore, let the reader be warned that many of the statements in this chapter may be on the verge of being blatantly false, when viewed by experts. If such falsehood is sensed, one of the more technical references should be consulted. A good place to start would be ftp://ftp.sas.com/pub/neural/FAQ.html . However, I maintain that in most applications, delving deeper in rigor amounts to polishing the proverbial cannonball.

[2] Variation on a theme by Arthur C. Clarke "Any sufficiently advanced civilization is indistinguishable from magic."

2.2 The Plan

It will help the reader to have a overview of this chapter. First, the emphasis is on regression and classification problems, where one is interested in predicting some quantity from another. In AI circles, the canonical tool for solving such problems is the neural network, or more specifically, the multilayered perceptron. I will utilize very simple and traditional (i.e., statistical) regression/classification models in order to "derive" neural networks. Consequently, the first half of the chapter is mostly basic statistics, with minimal mention of neural nets. When the multilayered perceptron does appear in Section 2.8, I make a point to maintain a statistical spirit. In other words, I emphasize statistical model building and its pitfalls. I will not delve into the intricacies of optimization (training) algorithms, for I think their impact on the final results is often over-stated, especially when one acknowledges that the results are subject to statistical uncertainty.

In this chapter, the span of techniques considered is sufficiently wide to include techniques that have more AI-ness to those which have higher levels of statistic-ness. I have attempted to provide computer code that implements most of the techniques examined. The codes are written in R, which is a computing environment similar to S or MATLAB, but it is freely available from http://cran.r-project.org. If the reader manages to work through the codes, chances are she will be able to tailor them to her own problems. If the code is not used at all, I hope the reader will leave this chapter, at the least, with a sense that many problems dealing with AI or statistics should be viewed in terms of distributions and histograms, or alternatives that acknowledge and convey uncertainty.

2.3 Dining Table – Regression

I measured the length of our dining table. It read 228.6 cm (nearly 90 in.). I then measured it again, and recorded 228.7 cm. To assure that the table was not perversely growing, I decided to measure its length $n = 100$ times, once an hour. This much of the story is mostly true, but the rest is not. I then, plotted a histogram of the measurements. It resembled that in Fig. 2.1. Clearly, the measurements varied quite a bit,

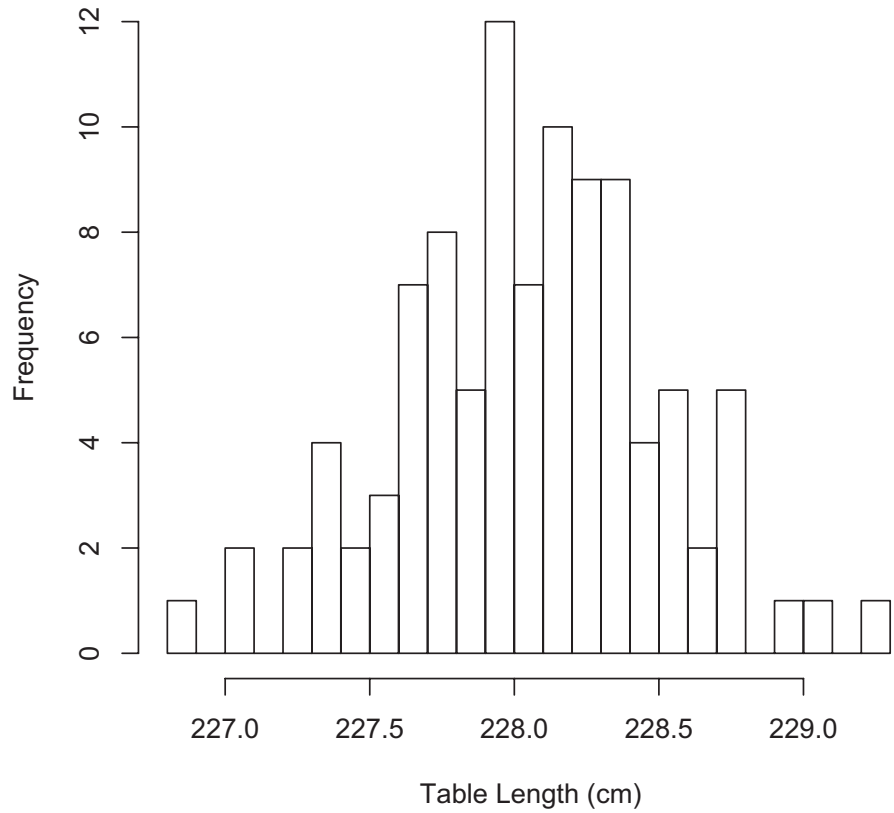


Fig. 2.1 The histogram of 100 length measurements

from a minimum of 226.89 cm to a maximum of 229.20 cm. The so-called sample mean and sample standard deviation of the length measurements were

$$\overline{y} = \frac{1}{n}\sum_i y_i = 227.98$$
$$s = \sqrt{\frac{1}{n-1}\sum_i (y_i - \overline{y})^2} = 0.517, \qquad (2.1)$$

where y_i denotes the ith observation of table length. A *point estimate* of the true length of the table is given by the sample mean, a measure of the center of the histogram. The standard deviation measures the spread of the histogram. (The quantity s^2 also has a name – sample variance.) Visually, the numerical values of both the sample mean and the sample standard deviation are consistent with the location and width of the histogram shown in Fig. 2.1.

I was almost ready to declare that the length of my dining table is 227.98 cm. Then I realized that the mean and standard deviation I computed are based on a sample of only 100 measurements. In other words, the true length of the table – called the population mean – may be something different. Basic statistics (Devore and Farnum 2005), teaches us that we can be about 95% confident that the true mean of my table is somewhere in the range $\overline{x} \pm 2\, s/\sqrt{n} = 227.98 \pm 2(0.517)/\sqrt{100} = (227.9, 228.1)$. This is called an *interval estimate* of the true length of the table. As such, on any time of day, I could *predict* the length of my dining table to be 227.98 ± 0.1. Now, that may not seem much of a prediction, but in fact, it is. It just happens to be a relatively simple prediction; a less-trivial prediction follows, next.

It then occurred to me that the table is next to a window where the daytime sun clearly shines on it for a good portion of the day. As such, the temperature of the table would have been varying from daytime to nighttime hours. Knowing a bit about physics and how things expand when heated, the natural question was how much of the $s = 0.517$ cm spread in my measurements of the table was due to changes in temperature? If I could quantify the contribution from temperature, I would be able to come up with a more precise estimate/prediction of the table length. To answer this

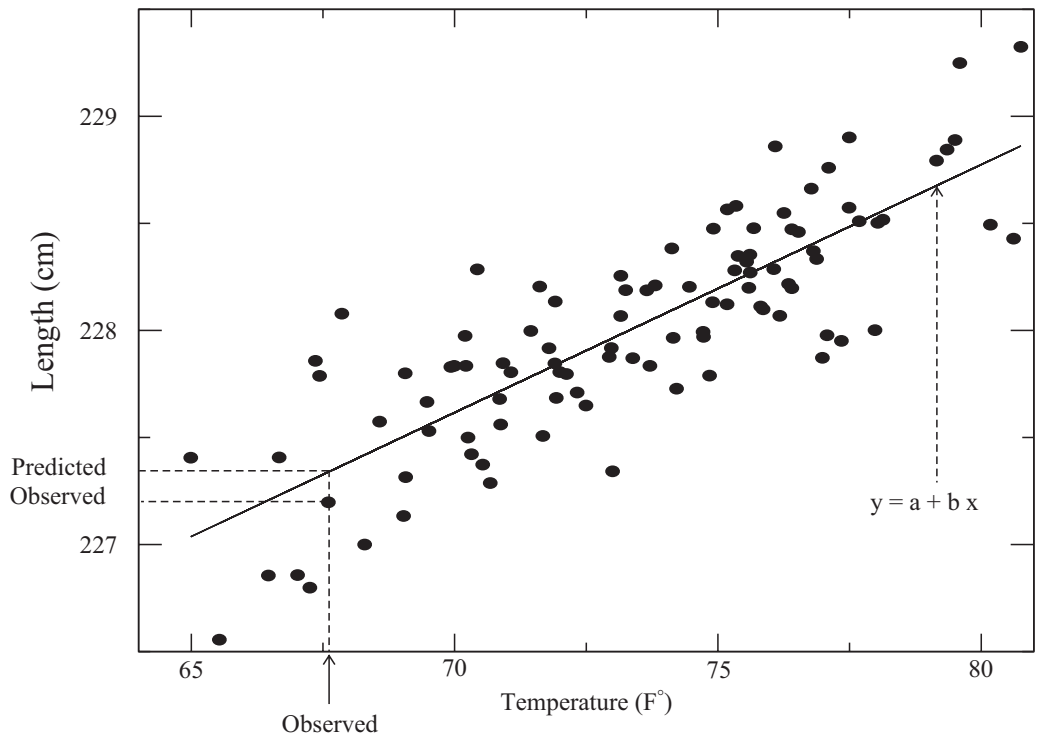


Fig. 2.2 Scatterplot of length and temperature for the 100 measurements, and the regression fit

question, I made another 100 measurements, but this time of both the table and its ambient temperature. I then plotted the numbers on what is called a scatterplot (Fig. 2.2).

The proper procedure for answering questions of the type "How much of the variance in one variable is explained by other variables?" is called regression (Draper and Smith 1998). One writes an equation such as

$$y_i = \alpha + \beta x_i + \epsilon_i, \tag{2.2}$$

where x_i and y_i are the ith measurement of temperature and table length, respectively. Of course, at this stage, one does not know the values of α and β, but let us assume for now that we know their values. The term ϵ_i represents an "error"; after all, it is unlikely that my data (x_i, y_i) fall exactly on a straight line $y = \alpha + \beta x$. If they did, I could claim that given the temperature x, I can predict the length of the table to be exactly $\alpha + \beta x$. So, ϵ_i is simply the difference between the actual measurement of the table length on the ith try (i.e., y_i) and what I would predict that length to be if there were a perfect linear relationship between temperature and length. Figure 2.2 shows the various elements of this discussion. In statistical jargon, the temperature is an example of a *predictor*, and the table length is called the *response*. In AI circles, they are called *input* and *target*, respectively. It is important to distinguish between the target, and the *output*; the former refers to the actual observed table length, while the latter refers to the predicted table length, $\alpha + \beta x_i$, according to the regression line (i.e., the straight line in Fig. 2.2).

Indeed, this way of interpreting ϵ_i, suggests a criterion for coming-up with values for α and β. Note that these parameters of the regression model represent the slope and y-intercept of the regression line. It is natural to ask what slope and y-intercept would give the smallest overall error. But there are n errors; how can one minimize all of them? Well, one cannot; but one can minimize some scalar function of the ϵ_i. A common choice is the Mean Squared Error (MSE). As the name suggests, it is defined as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} \epsilon_i^2 \ . \tag{2.3}$$

To see that this quantity is actually a function of the parameters, let us re-write it using equation (2.2):

$$\mathrm{MSE}(\alpha, \beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \alpha - \beta x_i)^2. \tag{2.4}$$

We can now minimize this MSE with respect to α and β using basic calculus;

$$\frac{\partial(MSE)}{\partial\beta} = -2\,\frac{1}{n}\sum_{i=1}^{n}(y_i - \alpha - \beta x_i)x_i, \tag{2.5}$$

$$\frac{\partial(MSE)}{\partial\alpha} = -\,\frac{1}{n}\sum_{i=1}^{n}(y_i - \alpha - \beta x_i). \tag{2.6}$$

The values of α and β which render these derivatives zero are called the Ordinary Least Squares (OLS) estimates, and are labeled $\hat{\alpha}$ and $\hat{\beta}$. The line with these particular estimates minimizes the mean squared error. According to equations (2.5) and (2.6), the following system of two linear equations with two unknowns ($\hat{\alpha}$ and $\hat{\beta}$) must hold true:

$$\overline{xy} - \hat{\alpha}\overline{x} - \hat{\beta}\overline{x}^2 = 0, \quad \overline{y} - \hat{\alpha} - \hat{\beta}\overline{x} = 0, \tag{2.7}$$

and so,

$$\hat{\beta} = \frac{\overline{xy} - \overline{x}\;\overline{y}}{\overline{x}^2 - \overline{x}^2}, \quad \hat{\alpha} = \frac{\overline{x}^2\;\overline{y} - \overline{xy}\;\overline{x}}{\overline{x}^2 - \overline{x}^2}. \tag{2.8}$$

Note that these OLS estimates of α and β can be computed from various averages computed from the sample data:

$$\overline{x} = \frac{1}{n}\sum_{i}^{n} x_i, \quad \overline{y} = \frac{1}{n}\sum_{i}^{n} y_i, \quad \overline{x}^2 = \frac{1}{n}\sum_{i}^{n} x_i^2,$$

$$\overline{xy} = \frac{1}{n}\sum_{i}^{n} x_i\;y_i, \tag{2.9}$$

For the data at hand, $\hat{\alpha} = 219.51$, $\hat{\beta} = 0.116$. It also turns out that a quantity called R^2 can also be computed from these averages; it expresses the percentage of the variance in the table length (i.e., 0.517^2) which can be attributed to the linear relationship with temperature (Draper and Smith 1998). For the table data, it is 68.5%. In short, the next time I want to predict my table length, I am better off to use the regression equation $219.51 + 0.116 \times$ (temperature), because it will give me a more precise estimate than the sample mean $\overline{y} = 227.98$.

Finally, if I suspect other culprits contributing to the variance in the length of my dining table – e.g., humidity, or how many books are piled on top – then I can include them in my *multiple* regression model as additional predictors, $y = \alpha + \beta_1 x_1 + \beta_2 x_2 + \ldots$.[3]

The moral of this story is that the parameters of the model (α, β) can be estimated according to some criterion, in this case the OLS criterion, directly from sample data (x_i, y_i), $i = 1, 2, \ldots, n$.

Now, in a book dealing with artificial intelligence and neural networks, why do we need to talk about estimating parameters? Because such parameters appear everywhere in neural nets – they are called weights – and even in some so-called non-parametric methods. Also what AI folk call "training" is nothing but parameter estimation; the sample data set employed for the estimation is called the "training set." Equally important, when we want to assess how well a model does, what we are really doing is estimating another population parameter – the prediction error. More on this, later. And why do we make such a big deal about the OLS criterion? Because the minimization of the MSE (i.e., the OLS criterion) is equivalent to the maximization of the probability of obtaining the data. More on this, next.

2.4 Basic Statistics – Parameter Estimation

In the previous section, I made the rather dramatic claim that if one minimizes MSE, then one has maximized the probability of obtaining the data. But what does that really mean? Surely, one cannot talk about the "probability of data" in complete generality. Indeed, one cannot. In order for claims like that to make any sense, one must have a *probability model* for the data. In this section, I will discuss two such models.

But, first: In the previous section I also referred to *population* and *sample*. These are two of the most important concepts in statistics. The former refers to

[3] The index i on these x's pertains to the different predictors, not the ith case of a single predictor, as in equations (2.2)–(2.9).

some "larger" set which for various reasons is often unavailable to us. What is available is only a sample taken from the population. A summary measure in the former is referred to as a (population) parameter, and one in the latter is called a (sample) statistic. For a data set consisting of a continuous variable – say, hourly temperature measured in centigrade, to 0.01 accuracy – an example of a summary measure is the mean. If the data are categorical, e.g., consisting of the occurrence or non-occurrence of some event – say a tornado – then a summary measure might be the proportion of the data that fall in each category. The business of parameter estimation is to infer something about the population parameter from sample statistic(s) computed from a single sample.

Often one describes the population in terms of an assumed probability distribution. For the purposes of this chapter, a probability distribution is a function that gives the probability of obtaining a single piece of data – an individual case. For all practical purposes, one can think of it as the histogram one would obtain (analogous to Fig. 2.1) for the entire population, if one had the entire population. Also often, these probability distribution functions have certain parameters appearing in various places in them. The parameters are usually introduced in strategic places in order to give them a natural interpretation. For the dining table example, one would arrange for one of the parameters in the distribution to represent the true (i.e., population) table length. Then, the question becomes if one can estimate that parameter from sample data? If so, then one has inferred the true length of the table from a single sample of observations.

To illustrate all this, let us consider two of the most famous distributions around, one for describing a categorical variable, and one for a continuous variable.

2.4.1 Binomial

Consider a coin, for which the probability of heads, labeled "1", is π. We all know that if we toss this coin n times, and if the tosses are independent (i.e., that the result of one toss does not affect the probability of another), then the probability of getting y heads in n tosses is given by the binomial distribution:

$$p(y;\pi) = \frac{n!}{y!(n-y)!}\pi^{y}\ (1-\pi)^{n-y}. \tag{2.10}$$

The quantity π is the parameter of the distribution.[4] As mentioned above, the choice of the term "parameter" to describe π is not accidental; indeed, we will use theoretical distributions like the binomial to describe the population, and as such, it is the parameter π which we would be interested in estimating. The notation on the left hand side of the equation denotes the probability of obtaining some value for the number of heads, y, if the data are described by a binomial distribution with parameter π. In other words, the equation tells us the probability of obtaining y heads, out of n tosses of a coin whose probability "heads" on each toss is π. Clearly, the possible values of y are $0, 1, 2, \ldots, n$.

Now, suppose we perform an experiment wherein the coin is tossed n times, and we find precisely n_1 heads. This, then, is our sample data. Can we estimate the value of the population parameter π? Most of us would not even think twice about our answer: n_1/n. But it turns out that that intuitive answer is actually based on the criterion that our estimate of π should maximize the conditional probability of obtaining the observed data, given the binomial model with parameter π. Observe! The probability of getting the data (i.e., n_1 heads out of n tosses), according to the model with parameter π, is

$$p(\text{data};\pi) = \frac{n!}{n_1!(n-n_1)!}\pi^{n_1}\ (1-\pi)^{n-n_1}. \tag{2.11}$$

To maximize this quantity we can simply differentiate it with respect to π, and set the result to zero. The algebra is considerably simplified if one instead maximizes the logarithm of the probability. I.e.,

$$\frac{\partial \log(p(\text{data};\pi))}{\partial \pi} = \frac{n_1}{\pi} - \frac{n-n_1}{1-\pi}. \tag{2.12}$$

The value of π at which this quantity is zero (labeled $\hat{\pi}$) is $\hat{\pi} = n_1/n$, namely the anticipated/intuitive answer.

The moral of this story is that an objective way for estimating a parameter of a population (or distribution) from a sample is to maximize the conditional probability of obtaining the observed sample, given some model for the distribution. This criterion is called the maximum likelihood criterion.

[4] n is also a parameter of the distribution, but for the sake of notational simplicity, only π is written on the left hand side of the equation.

2.4.2 Gaussian

A similar argument applies when y is a continuous variable. For example, y may have a Gaussian (or Normal) distribution. The parameters of the normal distribution are labeled μ and σ, and they appear in the distribution as follows[5]:

$$p(y;\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}(\frac{y-\mu}{\sigma})^2}. \tag{2.13}$$

For now, resist the temptation to call them mean and standard deviation; they are simply parameters of the distribution. As in the binomial case, the probability on the left hand side is the probability of obtaining an observation y, given that it is described by a normal distribution with parameters μ and σ.

Now, given a sample of size n, $y_i, i = 1, 2, 3, \ldots, n$, can we estimate the parameters of the distribution which describes the population? If the probability of getting any individual y is given by equation (2.13), and if the observations are independent, then the probability of getting the whole data set is given by the product

$$p(\text{data};\mu,\sigma) = \Pi_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}(\frac{y_i-\mu}{\sigma})^2}. \tag{2.14}$$

What are the values of the parameters that maximize this probability? Let us call them $\hat{\mu}$ and $\hat{\sigma}$. Again, to find them we just maximize the logarithm, and set the result to zero. But now we have two parameters, and so we must set both derivatives equal to zero:

$$\frac{\partial}{\partial\mu} p(\text{data};\mu,\sigma)|_{\hat{\mu},\hat{\sigma}} = 0 = \frac{\partial}{\partial\sigma} p(\text{data};\mu,\sigma)|_{\hat{\mu},\hat{\sigma}}. \tag{2.15}$$

These two equations yield

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} y_i, \quad \hat{\sigma^2} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\mu})^2. \tag{2.16}$$

One can see now why the parameters μ and σ may be called the mean and the standard deviation of the distribution (or the population) – because they are estimated with the mean and the standard deviation of the sample; see equation (2.1).[6] So, just as in the case of the binomial, the maximum likelihood criterion can be employed to estimate population parameters.

2.4.3 Back to Dining Table

What is a probability model for my dining table? In other words, what is the analog of equations (2.10) and (2.13)? These equations pertain to a single variable we call y, but the dining table example involves two variables – y, the length of the table, and x, the temperature. It turns out that a probability model is given by equation (2.13), but with the parameter μ replaced by a function of x, denoted $\mu(x)$ [7]:

$$p(\text{data};\mu(x),\sigma) = \Pi_i^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}(\frac{y_i-\mu(x)}{\sigma})^2}. \tag{2.17}$$

The specific choice $\mu(x) = \alpha + \beta x$ is the one we adopted in equation (2.2) when we were attempting to find a linear relationship between temperature (x) and length (y). In fact, for this specific linear model, equation (2.17) becomes

$$\begin{aligned} p(\text{data};\mu(x),\sigma) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}\sum_i^n \left(\frac{y_i-\mu(x)}{\sigma}\right)^2} \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}\sum_i^n \left(\frac{y_i-\alpha-\beta x_i}{\sigma}\right)^2} \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2}nMSE(\alpha,\beta)}, \end{aligned} \tag{2.18}$$

where $MSE(\alpha, \beta)$ was defined in equation (2.4).

We are finally prepared to make sense of the statement made at the end of the previous section: Maximizing the probability of the data, i.e., the left hand side of equation (2.18), is equivalent to minimizing the exponent on the right hand side, i.e., MSE. A more elegant way of saying this is as follows: for a Gaussian probability model (i.e., when the errors,

[5] I am a bit cavalier in referring to this quantity as a probability distribution. When y is a continuous variable, $p(y;\mu,\sigma)$ is not the probability of y. Technically, it is a probability *density*, which means that its integral (or the area under it) between two limits represents probability. After all, the expression given for $p(y;\mu,\sigma)$ is not even restricted to the $0-1$ interval. So, to be more rigorous, every $p(y)$ in this chapter should be multiplied by a dy before interpreting the product of the two as a probability.

[6] From equation (2.1), we see that s^2 has an $(n-1)$ in its denominator, while $\hat{\sigma^2}$ in equation (2.16) has an n in its denominator. But for sufficiently large n, that difference is small anyway.

[7] Note that the parameter σ is assumed to be a constant, independent of x. In regression circles, this is referred to as the assumption of homoscedasticity.

ϵ_i are normally distributed), the maximum likelihood estimate of the population parameter is equal to the OLS estimate (equation (2.8)).

In passing, let us note that, motivated by equation (2.16), we call μ the mean of y in the population. A calculation similar to that leading to equation (2.16) suggests that $\mu(x)$ is in fact the *conditional* mean of y. In other words, the output $\alpha + \beta x$ is the mean of the response y, for a given value of the predictor x. This is exactly what we would want as the output of a model – the mean of the response, given the predictor.

2.4.4 Estimation Criteria

We have seen how the business of estimating the parameter of some probability distribution can be cast into the problem of minimizing some error (or cost, or loss) function. For example, in simple linear regression, to estimate the parameter, $\mu(x)$, of the Gaussian distribution, we minimize the mean-squared error $(1/n)\sum_i (y_i - \mu(x))^2$; see equations (2.5) and (2.6). We called the resulting parameter estimates OLS estimates. However, we might have simply decided that the OLS estimates are what we consider to be relevant, without any attention to the connection with the underlying probability distribution. Of course, what will be lost in that case is the interpretation of the outputs as a conditional mean (see end of previous section), but so what? Perhaps, we do not care about that interpretation, and all we want is a regression model that makes predictions which are "optimal" or "best" according to some non-probabilistic measure.

The point is that one can adopt any criterion for parameter estimation, independently of a probability model. And that is one place where AI and statistics have some differences; although certainly many statisticians have in the past chosen to estimate parameters by minimizing some error function regardless of the connection with a probability distribution, one can justifiably give more of the credit (or the blame) for that practice to AI folk.

Either way, let us see how some popular criteria do lead to sensible estimates. For purposes of illustration, consider a single variable y. Suppose we have some data on this variable: $y_i,\ i = 1, 2, 3, \ldots, n$. Now, let us ask what is the "best" prediction/forecast f we can make for the next value of y. For instance, y may be daily temperature highs for January 1, and so, y_i label all the highs for n years. Based on this data, what should be our prediction for next year's temperature high on Jan. 1?

The notion of "best" is quantified by adopting some criterion. Suppose we adopt the criterion that the forecast should minimize the mean squared error, i.e.,

$$\frac{1}{n}\sum_i^n (y_i - f)^2. \tag{2.19}$$

Simply differentiating this expression with respect to f, and setting the result to zero, yields the "best" forecast

$$f = \frac{1}{n}\sum_i^n y_i, \tag{2.20}$$

which we recognize as the sample mean of the data. So, the criterion of minimizing the mean squared error, is tantamount to estimating f by the sample mean of y.

Another interesting criterion is to minimize the mean absolute error, i.e.,

$$\frac{1}{n}\sum_i^n |x_i - f|^1. \tag{2.21}$$

It is possible to show that the f that minimizes this quantity is the median of the data. With a little more work one can also show the curious, but not very useful, result that the forecast which minimizes

$$\frac{1}{n}\sum_i^n |x_i - f|^\alpha \tag{2.22}$$

as $\alpha \to \infty$, is the midrange $(x_{min} + x_{max})/2$, if the x-values are restricted to the range $[x_{min}, x_{max}]$.

All of these criteria deal with the estimation of a single quantity (f). But the results generalize to the case where f is itself a function of some predictor, if the word "conditional" is inserted in the right place. For example, minimizing the mean absolute error in regression, assures that the output represents the conditional median of the response, i.e., the median of y, for a given x. The next section gets deeper into the intricacies of conditioning.

2.5 Regression and Classification

The whole business of regression and classification is about estimating the relationship between two sets of variables from data. Depending on the field, the two sets of variables are called by different names. In this

chapter, I will refer to one set as input variables, inputs, or predictors; the other set includes output variables, outputs, target variables, or response. As mentioned in Section 2.3, one important distinction should be made within the latter set: The output of regression is not necessarily the target, but some approximation to it. This can be seen in Fig. 2.2, where the y-values of the regression line (i.e., its outputs) are not identical to the y-values of the points (i.e., targets). In a regression problem, the map or the function of interest is the one that literally maps the inputs to the targets in some optimal sense. The straight line in Fig. 2.2 is one example. In a classification problem the analog is the *(decision) boundary* that best demarcates the regions occupied by the various classes. Look ahead at Fig. 2.8 for an example of a nonlinear boundary between two classes.

As advertised in the introduction section, central to regression and classification problems is the conditional probability, $p(Y = y|X = x)$, the probability of Y taking some value y, *given* that X takes some value x. If Y is continuous, it tells one the probability of any y value, given some x value. The expression in equation (2.17) is an example of such a probability. And if Y is categorical, then, for example, $p(Y = 1|X = 10)$ could be the probability of a tornado occurring, if wind speeds happen to be 10 knots.

Let us now explore regression and classification problems more closely. This will set the stage for the neural network.

2.5.1 Linear Regression

We have already dealt with linear regression in Sections 2.3 and 2.4.3 of this chapter, but there is more to know. Regression models come in two types: linear, and nonlinear. It is important to realize that those adjectives refer to the parameters of the model, and not the relationship between the input and output variables. A model like $y = \alpha + \beta x^2$ is really a problem in linear regression, in spite of the quadratic nature of the relation between x and y. One way to see why, is to note that one can simply square the values of x in the data, and then do linear regression between y and x^2. The other way is to note that the equation is linear in the parameters α, β. This implies a unique solution that can be obtained via the OLS criterion; see equation (2.8). A truly nonlinear regression model would be something like $y = e^{\beta x} + \alpha$. The corresponding expression for mean squared error for such a problem is nonlinear in the α, β parameters, and so cannot be minimized simply. In such a case, one is truly in the realm of nonlinear optimization where unique solutions for the parameters often do not exist. As we shall see, neural nets fall into this category.

Let me mention in passing that a given problem may be treated as a regression or a classification problem. For example, suppose we are interested in predicting wind speed from knowledge of pressure difference, and whether or not clouds are present. This could be cast as a regression problem with wind speed as the response; the predictors would be pressure difference and the presence/absence of clouds. But one can also encode wind speeds as "slow", "medium", or "fast". Then, we can treat the problem as a classification problem. One may wonder what type of problem – regression or classification – one is dealing with if the target variable is categorical, but with a large number of categories, e.g., the thousands of letters appearing in the Chinese language. Given the large number of classes, one may be tempted to treat this as a regression problem, but in fact, because the classes have no inherent order, it is much more suitable as a classification problem. Although it is not always obvious whether one should treat the problem as regression or classification, if the target is not ordinal (i.e., if the classes have no inherent order), then classification is the better way to go. And that is the topic of the next section.

2.5.2 Discriminant Analysis

A traditional classification model, called Discriminant Analysis, is an enlightening ancestor of neural nets. Let us look at the simple case of one input variable, x, and two classes labeled as $i = 0, 1$ (e.g., non-tornado, and tornado). The model begins by assuming that the likelihood of x, when it happens to belong to class i, is Gaussian. In other words, the conditional probability of x, given class i, is written as

$$L_i(x) \equiv p(x|C = i) = \frac{1}{\sqrt{2\pi\sigma_i^2}} \exp^{-\frac{1}{2}(\frac{x-\mu_i}{\sigma_i})^2} \tag{2.23}$$

where μ_i and σ_i are the mean and standard deviation of x when it belongs to the ith class. Note that $L_0 + L_1 \neq 1$.

For forecasting purposes, this likelihood is the wrong probability to look at. We are not interested in the probability of finding, say, wind speed = 50 km/h, given that there is a tornado on the ground. We want to know the probability that there is a tornado on the ground, given that wind speed = 50 km/h. This other probability, $p(C = i|x)$, is called the posterior probability, and the laws of probability imply that it is given by

$$P_i(x) \equiv p(C = i|x) = \frac{p_i L_i(x)}{p_0 L_0(x) + p_1 L_1(x)}, \quad i = 0, 1, \tag{2.24}$$

where p_0, p_1 are called prior (or climatological) probabilities for the two classes. These prior probabilities are independent of x, and are usually estimated with N_0/N, and N_1/N, respectively, where N_i is just the sample size of the ith class, and $N = N_0 + N_1$ is the total sample size. Note that $p_0 + p_1 = 1$ and $P_0 + P_1 = 1$.

When we observe some value of x, if $P_1(x) > P_0(x)$, then we forecast (or classify) it as a 1 (i.e., a tornado). Actually, instead of $P_1(x) > P_0(x)$, it is more convenient to look at $\log(P_1(x)/P_0(x)) > 0$, because we can then write the left-hand side as

$$\log(P_1(x)/P_0(x)) = \frac{1}{2} D^2(x), \tag{2.25}$$

where the so-called discriminant function, $D^2(x)$, is given by

$$D^2(x) = \left(\frac{1}{\sigma_0^2} - \frac{1}{\sigma_1^2}\right) x^2 - 2\left(\frac{\mu_0}{\sigma_0^2} - \frac{\mu_1}{\sigma_1^2}\right) x + \left(\frac{\mu_0^2}{\sigma_0^2} - \frac{\mu_1^2}{\sigma_1^2}\right) + 2\log\left(\frac{\sigma_0}{\sigma_1}\right) - 2\log\left(\frac{1 - p_1}{p_1}\right). \tag{2.26}$$

This equation follows in a straightforward way by combining equations (2.23)–(2.25). Note that this discriminant function is equivalent to the decision boundary discussed at the beginning of Section 2.6, except that there we were considering two predictors. The point is that by setting the discriminant function to a constant, and solving the resulting equation for the predictor(s), leads to the equation for the decision boundary. In this one-predictor example, a decision boundary simply refers to one or two values (i.e., thresholds) of x, namely the zeroes of the discriminant function (2.26).

"Training" in discriminant analysis refers to estimating the means and standard deviations appearing in equation (2.26). Consequently, training refers to estimating some optimal value for the above-mentioned thresholds. Again, an observation x from either training set or an independent set is forecast as (or assigned to) class 1 (e.g., tornado), if $D^2(x) > 0$, otherwise it is classified (forecast) as a 0 (e.g., non-tornado).

It can be seen from equation (2.26) that if the variances of the two classes are equal, then the discriminant function is linear, and the posterior probability, $P_1(x)$, in equation (2.24) becomes a smoothed step function:

$$P_1(x) = \frac{1}{1 + exp^{-(\alpha + \beta\, x)}}, \tag{2.27}$$

where $\alpha = \frac{\mu_0^2 - \mu_1^2}{2\sigma^2} - \log(\frac{1-p_1}{p_1})$, and $\beta = \frac{\mu_1 - \mu_0}{\sigma^2}$. Functions of the type (2.27) are called logistic and will be seen again, below, in dealing with neural networks. For now, just note that the logistic function arises naturally in dealing with linear discriminant analysis. Meanwhile, with unequal variances one allows for a quadratic discriminant function. In that case, $P_1(x)$ can behave more non-trivially. Figure 2.3 shows the above ingredients in both the linear and quadratic case.

In spite of its simplicity, discriminant analysis is a very powerful classification model. But as seen from equation (2.26), the best it can do is to handle quadratic decision boundaries. It will not do well if the boundary underlying the data is more nonlinear than that. Of course, the source of this limitation is the starting assumption – normality, equation (2.23). However, there exist classification methods where the assumption of normality is either dropped or relaxed, and in those methods the decision boundaries can be much more general. Neural nets constitute one way of addressing this distributional constraint. They allow for almost any distribution, and as a consequence, they allow for highly nonlinear decision boundaries. Even if you never see or hear about discriminant analysis again, one lesson that should be learned from it is that the distribution of the data is related to the nonlinearity of the decision boundary. In my personal experience, discriminant analysis is also a worthy competitor of neural nets, and so I strongly urge the reader to develop it along side a neural net.

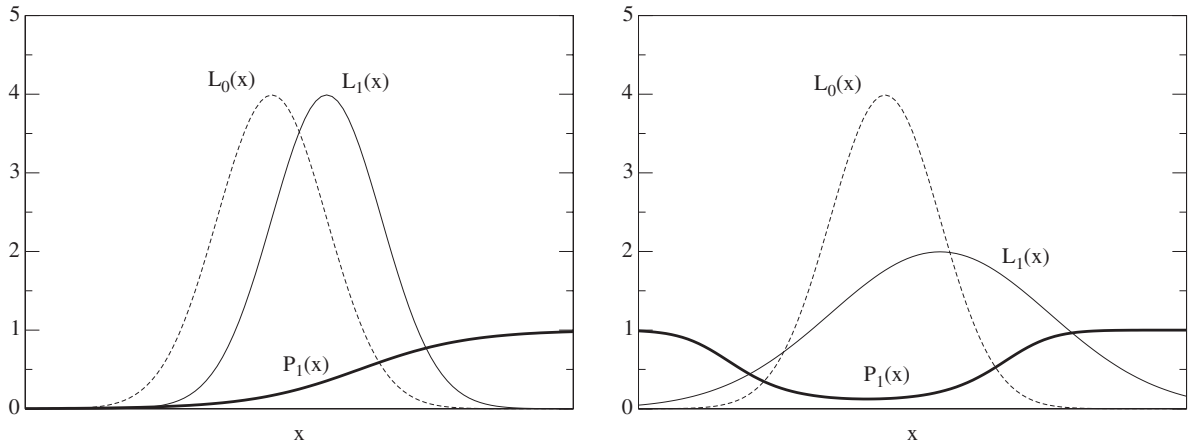


Fig. 2.3 The ingredients of discriminant analysis. The likelihoods for each of two classes (L_0 and L_1), and the posterior $P_1(x)$ for linear (left) and quadratic (right) discriminant analysis

2.5.3 Support Vector Machines

At the start of Section 2.5.1, I mentioned a trick of taking a problem that looks nonlinear, and linearizing it. There is one more lesson that can be learned from that trick. Let me, first, restate it. A linear regression model is $y = \alpha + \beta x$, and it is an appropriate model if the relationship between x and y is linear. What if the relationship is not linear, say, quadratic? Easy: the appropriate linear regression model is then $y = \alpha + \beta_1 x + \beta_2 x^2$. The model is still linear because it is linear in the parameters α, β_1, β_2, but the relationship between x and y is now nonlinear. Continuing this line of thinking, it becomes evident that a polynomial of order P, with sufficiently large P, can accommodate (almost) any amount of nonlinearity in the data. Note that the terms in such a polynomial can be treated as a different predictors/inputs in a multiple regression model.

Cortes and Vapnik (1995) asked themselves how many inputs it will take to separate a two-class data set with a second-order polynomial boundary. On the one hand, we can simply employ p inputs of the form $x_1, x_2, \ldots, x_p$, and allow a nonlinear model to map those to a binary output. On the other hand we could use a linear model that in addition to the p inputs also includes another p inputs of the form $x_1^2, x_2^2, \ldots, x_p^2$, plus another $p(p-1)/2$ inputs of the form $x_1x_2, x_1x_3, \ldots, x_px_{p-1}$. After taking symmetry into account, that amounts to a total of $p(p+3)/2$ inputs, feeding into a linear classifier.

As such, the data is mapped to a larger-dimensional space where the decision boundary is a linear hyperplane. This is the main idea behind Support Vector Machines (for classification) and Support Vector Regression, the latter described in Chapter 7. One criterion for selecting a hyperplane fitting through the data is to simply pick the one that maximizes the geometric margin between the vectors of the two classes. The main advantage of this choice is that it requires only a portion of the training set, namely those closest to the hyperplane, often referred to as support vectors.

I will not pursue this topic much further. The only reason I mention it here is to demonstrate that even a technique that most people would consider an AI technique, is firmly founded in traditional notions of regression and classification.

2.6 Model Selection and/or Averaging

We are now in position to discuss a large topic, one that effects many statistical models – traditional or otherwise. Consider a multiple regression model involving one output and multiple, say, p, inputs: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots \beta_p x_p$. We already know how to estimate the β_i parameters from a training set. The equations are analogous to those given in equation (2.8) for the case $p = 1$. But how do we decide what the best value of p is? Even worse, suppose the regression model we are trying to fit to the data is a polynomial

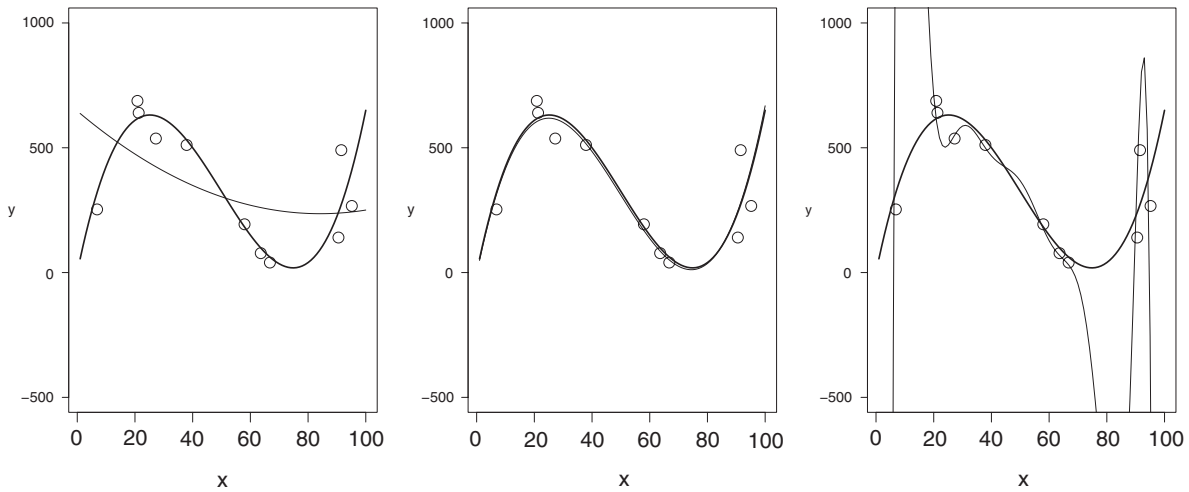


Fig. 2.4 Examples of an underfit (left), a good fit (middle), and an overfit (right)

of some order, in multiple predictors. How many predictors, which predictors, and what order polynomial should we allow in our model? In other words, we can train a model with a given structure, but how do we decide on the structure itself? The body of knowledge developed to address that problem is called Model Selection.

Why do we need to worry about model selection at all? Why can we not throw every possible predictor in a huge-order polynomial regression model, and not worry about which number of predictors and what order optimal? Indeed, we could, if we had infinite data. But with a finite sample size, we cannot, because of something called overfitting.

Consider the situation where we have 11 cases in our data, and just suppose the two variables are related via a cubic equation. Figure 2.4 shows an example of such data (i.e., the circles). In fact, I can tell you that the underlying relationship is indeed cubic, because that is how I made this data: I took the line $y = 0.01x^3 - 1.5x^2 + 56.5x + 0.01$ and added some noise to the y.[8] The dark solid line shows the cubic relation itself. What would happen if I fit a second-order polynomial to this data? The left panel in Fig. 2.4 shows the resulting fit. It is clearly not capturing all the features of a cubic relation, and as such, it is underfitting the data. How about a cubic fit? The middle panel shows that one. It looks good, and indeed the estimated parameters from this sample are 0.01007, −1.50207, 55.99351, −6.10929 – pretty close to the true (population) values. So, a cubic fit actually comes very close to capturing the true underlying relation. In other words, if you take this cubic fit and make predictions with it on *new* data from the population, independent of the 11-case training data, it will do well.

Finally, what if I were overly generous with the order of the polynomial and decided to fit the data with a 10th-order polynomial? The result is the curvy line in the right panel. Clearly, this is not the right fit – it is called an overfit. Overfitting occurs because the model has too many parameters, compared with the size of (and noise in) our data. But what is wrong with this overfit anyway? What is wrong is that it will produce nonsensical predictions when it is tested on new data. Just consider the predictions (i.e., the y-values) in the range $x = 10$ to $x = 20$; the predictions should be near the dark line, but our polynomial model is predicting values literally off the chart. If you were predicting temperature from wind speed, the forecast would be for an exceedingly hot day (and my dining table would not fit in the dining room).

One may object to my argument by pointing out that the region with x in the 10 to 20 range is a particularly data-sparse region, and that we should expect

[8] Here is the R code that made the data, in case the reader would like to play with it:

```
set.seed(1)
x = 100*runif(11, 0, 1) # select random x
y = 0.01 * x^3 - 1.5 * x^2 + 56.5 * x + 0.01 +
    rnorm(11, 0, 40) # add normal noise to polynomial
```

the predictions to be bad in that region. But, consider the region with $x \sim 90$; it is a data-dense region, and yet the predictions vary violently from very large y-values to very small y-values. So, however one looks at this model, it is a bad fit, and we would lose money using it.

2.6.1 Hold-Out and Resampling Methods

We ended the previous section by noting that an overly simple model will underfit the data, and an overly complex one will overfit. In practice, of course, we do not know what the underlying relationship is, and we cannot even plot the data on a scatterplot (like Fig. 2.4), because we may have tens or even hundreds of predictors, instead of the single x in the above discussion. So, instead of looking at the fits, one simply computes some measure of how close the fit comes to the data; for example, mean squared error, $(1/n)\sum(t_i - y_i)^2$, where t_i are observed y-values, and y_i are the corresponding predictions from the fit. The mean squared errors for our three fits are 34700.76, 8668.13, and 0, respectively. So, we could select a model based on this training error (also called the *apparent error*, when the training set is the entire sample); but the problem is that this training error is guaranteed to fall off with increasing model complexity (e.g., order of the polynomial). So, if we select a model with the lowest training error, we are likely to select a model that overfits the training data, and will therefore perform poorly on new data.

To avoid this problem, one often divides the data into two parts – a training set and a validation set. The first is employed for estimating the parameters of the model (just like we did above), and the second set is used for a variety of purposes. If the validation set is reasonably large and representative of the population, then one can legitimately claim that the value of the error on the validation set is a reasonable estimate of the error if the model were to be applied to the population. The latter is called the *prediction (or generalization) error*, and estimating it is the business of *model assessment*. One can use a large validation set also for model selection, by simply selecting the model that has the best performance on the validation set. In addition to training and validation sets, one often reads about a third "test set". In those situations, the training set is used for estimating the parameters of the model, the validation set is used for selecting the best model, and the test set is for obtaining an unbiased estimate of the prediction error. But with the methods described next, there is no need for a test set.

The problem sets in when the validation (or test) set is small. Consider model selection: In the so-called *hold-out* scheme, one trains several models – with varying levels of complexity – on the training set, and then selects the model with the lowest error on the validation set. This may make sense, because it certainly looks like we are avoiding overfitting the training set. However, the flaw in this scheme is that the model that has the lowest error on the validation set is now overfitting the validation set. After all, there is no reason to expect future data to be anything like the single validation set on which the model does best, especially if the validation set is small. As such, the model selected in this way is still not the best model to use on future data.

Having taken the step of dividing the data into two sets – training and validation – it is natural to go one step further. If we want to avoid overfitting a given training and a given validation set, why not have several training and validation sets, say 1,000 training sets, and 1,000 validation sets? Note that I am not talking about a training set of size 1,000, but 1,000 different training sets, each of some size. If we can somehow devise a scheme that utilizes a number of different training sets and a number of different validation sets, then there is little chance of overfitting anything. And where will we get 2,000 different training and validation sets? Data does not usually grow on trees. Here, I will discuss two tricks of the trade – cross-validation and bootstrapping – without arguing for or against either one. In practice, one should explore both of them, if one has that luxury, because they tell different stories and have their own pros and cons (Kohavi 1995).

It is important to point out that these methods can be used for model assessment and model selection. The emphasis in the former is to obtain a good estimate of some population parameter, for example the prediction error. In other words, we want to know how well a given model will perform on the population at large. The second purpose is to decide which model is better. Of course, we can always obtain estimates of the prediction error for each of the models, and then pick the model with the lowest prediction error. That would be

only one criterion for model selection. Other criteria may not involve the prediction error at all.

One way to generate multiple training and validation sets from a single data set is to divide-up the data into several chunks, say k, train on $k-1$ of the chunks and validate on the unused kth chunk. This way we can have k different training and validation sets. This method is the basic version of what is generally called cross-validation (Stone 1974). The case with $k = 2$ is known as the split-sample method. At the other extreme, when k is as large as the sample size itself, the method is called leave-one-out cross-validation, for obvious reasons. For the theoretically-inclined reader, Zhang (1992), and Shao (1993, 1996) show that leave-one-out cross-validation is asymptotically inconsistent, which means that as the size of the sample approaches infinity, the probability of selecting the true model (in a predictive sense) does not approach 1. I will not go into its details here; Fu et al. (2005), and Zucchini (2000) provide a thorough discussion of the nuts and bolts of cross-validation, at large.

Another way of producing multiple training and validations sets is to appeal to resampling. In resampling methods (Cherkassky and Mulier 1998; Hastie et al. 2001; Hjorth 1999; Masters 1995) one typically takes a sample from the entire data set and uses it for training. The remainder of the data can then be treated as a validation set. Given that the training set is drawn randomly from the sample, we can always take lots of different samples from a given data set, and that is how we are going to collect our 1,000 training sets. In fact, we can resample ad nauseum, unlike cross-validation where a maximum of k training sets can be drawn. Two questions that arise are (1) how large should each sample be, and (2) should the sampling be done without or with replacement? Intuitively, the advantage of sampling without replacement is that a training set does not end-up having multiple copies of the same case. The disadvantage is that a training set will end-up being smaller than the whole data set. This disadvantage is not too important in the context of model selection, but it is important if one is using the resampling method for estimating the prediction error. In the model selection context, sampling without replacement has been employed in the literature, but not enough to have earned its own name. One can describe it as hold-out cross-validation with random subsampling (Kohavi 1995).

When the sampling is done with replacement, then one can draw a training set as large as the original sample itself. As mentioned above, this is important in the context of model assessment, because one generally desires to produce an interval estimate for some parameter (e.g., prediction error), and for a given sample size. In resampling lingo, a sample taken with replacement, as large as the data itself, is called a *bootstrap* sample (Efron and Tibshirani 1998), and the number of times the resampling is performed is called the number of bootstrap trials. The most crude estimate of the prediction error involves averaging the training error over all the boostrap trials. This average training error, as we have been saying repeatedly, constitutes an optimistically biased estimate, because a larger network will generally lead to a lower value of this error. A better estimate (Efron and Tibshirani 1997) is called the leave-one-out bootstrap estimate, and is denoted $Err^{(1)}$:

$$Err^{(1)} = \frac{1}{N}\sum_{i}^{N}\frac{1}{|B^{-i}|}\sum_{b\in B^{-i}} L_{i,b}, \tag{2.28}$$

where N is the sample size, B^{-i} is the set of bootstrap trials that do not include the ith case in the data, and $L_{i,b}$ is the error committed on the ith case, by a model trained on the bth bootstrap sample. Note that $Err^{(1)}$ averages over the bootstrap trials, first, and then over the cases. Two common choices for $L_{i,b}$ are given in equations (2.32) and (2.33), specifically in the argument of the sum. $Err^{(1)}$ is better because it does not refer to a single training or validation set. However, it has been shown that the resulting estimate of the prediction error is, in fact, pessimistically biased. In other words, it over-corrects for the faults of the basic bootstrap estimate. To correct for that over-correction, Efron (1983) and Efron and Tibshirani (1997) propose what is called the 0.632 estimate. The name originates from the fact that in the long run only $(1-\frac{1}{e}) = 63.29\%$ of the cases in the original sample are utilized for training. The conclusion of the analysis is that a pretty decent estimate of the prediction error is given by the following weighted average of the apparent error and the leave-one-out bootstrap estimate:

$$\text{Prediction Error} = 0.368 \times \text{Apparent Error} + 0.632 \times Err^{(1)} \tag{2.29}$$

where,

$$\text{Apparent Error} = \frac{1}{N}\sum_{i}^{N} L_i \tag{2.30}$$

and L_i is the error committed on the ith case in the data, by a model trained on the entire data. All sorts of variations on this theme have been explored: The 0.632+ method (Efron and Tibshirani 1997), out-of-bootstrap (Rao and Tibshirani 1997), and bagging (Breiman 1996), as well as numerous others are nicely discussed in Hastie et al. (2001).

The estimate I use for the example illustrated in Section 2.8.3 is the one given in equation (2.29). To summarize, given a data set of size n, I draw a sample (with replacement) of size n, treating it as a training set, and the remainder as the validation set. A number of models are trained on the training set, and the performance of each model on the respective validation set is recorded. The sampling procedure is repeated many times, say 1,000. As such, 1,000 models are trained on 1,000 training sets, and validated on 1,000 validation sets. $Err^{(1)}$ is computed according to equation (2.28), and the apparent error is obtained by training the model one time on the entire data set. These two are used in equation (2.29) to estimate the prediction error. The model with the lowest prediction error is expected to perform best in the population, i.e., has the best generalization performance (in AI lingo).

Now, note that all of this resampling has gone to produce a single point estimate of the prediction error. Although that may suffice for many applications, it does not when model selection is the task. To properly select the best model, we need an interval estimate of the prediction error. After all, the prediction error is a random variable, and therefore, a point estimate of prediction error is subject to variability. In deciding on a good model, it is important to have some sense of the variability of the prediction error. Efron and Tibshirani (1997) report some analytic formulas for computing the standard error of prediction error, but the formulas do not apply to all performance measures. That is not a problem, because one can again rely on resampling techniques to estimate the variability of the prediction error. One can justifiably refer to this approach as a double-bootstrap, where one (inner) bootstrap is done to obtain a point estimate of the prediction error, and then another (outer) bootstrap is done to get the empirical sampling distribution of that estimate. The idea of the double-bootstrap was introduced by Efron (1983), but the specific application of double-bootstrap for model selection has been addressed, most recently, by Hall and Maiti (2006), and Tian et al. (2007). As simple as that idea seems to be, its proper formulation is subtle and complicated. Here, I will not get into those details. Instead, I will illustrate what may be called the "poor man's double bootstrap." What that entails is simply taking multiple samples (without replacement) from the data, and then computing a point estimate of the prediction error for each sample via proper bootstrapping. Then histograms or boxplots of prediction errors will give us a sense of the variability of the prediction error.

One last technical (but important) point remains. Suppose we have completed this type of double-bootstrap and have obtained two boxplots of prediction errors, one from model A, and another for model B. Can we simply compare these boxplots by placing them side-by-side and noting their relative position? For example, if there is significant overlap between the two boxplots, can we conclude that the two models are statistically equivalent? As tempting as it may be to answer in the affirmative, that answer would be correct only if we were to assume that the data that go into making the boxplot for model A are independent of the data that go into making the boxplot for model B. In the scheme outlined above, that assumption is violated, because each point estimate of prediction error, for model A and model B, is based on the same data set! This happens because often one takes a sample from the data set – the outer bootstrap sample – and then computes the prediction error – via the inner bootstrap – for model A and model B. As such, the values of the prediction error for model A and model B are not independent across the outer bootstrap trials. In statistics they are said to be "paired data." In such a situation, one examines the boxplot of the *difference* of the prediction errors from model A and model B. Then the position of this boxplot, relative to zero, is a measure of whether or not the two models are statistically equivalent. Suppose the difference is taken as "model A – model B". If the boxplot of this difference in the prediction errors is completely above zero, then we may conclude that model B has a statistically lower prediction error than model A. Similarly, model A has a statistically significant lower prediction error than model B, if the boxplot of the difference is entirely below zero. And finally, if the boxplot includes zero,

then one cannot conclude that one model is better than another, at least not based on the data at hand.[9]

All of this methodology for model selection is completely general. It is not specific to finding the optimal order of a polynomial regression; it may be used to select the right number of inputs into some model. Quite generally, it can be used to determine the optimal structure of any model, given data. It is unfortunate that the methodology does not tell us how much data we need to prevent overfitting, but it does tell us whether or not we are overfitting the data at hand. In Section 2.8.3, the above methodology is applied to a concrete example in order to address the complexity of a neural network model.

2.6.2 Right Structure but Now What?

Suppose we have utilized this model selection method to select the order of a polynomial regression model. But which one of the 1,000 models with this specific order should we use when we want to make predictions on new data? There are at least two answers: (1) Take all 1,000 polynomials, and average their predictions to obtain a single prediction. This idea is based on what Breiman (1996) calls bagging (for bootstrap aggregation), an example of a larger class of techniques called model averaging.[10] It is nice because it can also give us a sense of the spread/uncertainty in the forecast. (2) Alternatively, one can take a model with the optimal order, as suggested by the model selection method, and retrain it on the entire data. This option yields a single model that can be used on new data. In my experience, either one is good enough – and better than the hold-out method, especially when one has small validation sets.

Bagging is even more general than the manner in which I presented it. Theoretically, the advantage of bagging is that one does not have to worry about model complexity at all! In other words, you can set the order of the polynomial to some huge number, and proceed with its training on the bootstrap training sets. Of course, each polynomial model trained on a given bootstrap sample may overfit the training set, but each polynomial will overfit a different sample. The idea is that the average of all these different overfit models is itself a model that does not overfit.

2.7 Nonparametric Models

The regression equation $y = \alpha + \beta x$ is called a parametric model of data, not because of the parameters α and β in the equation, but because it cannot fit a wide range of relationships. A nonparametric statistical model generally refers to an algorithm that has sufficient flexibility to capture a wide range of relationships in the data. As such, polynomial regression is a nonparametric model, in spite of the abundance of parameters in it. Its flexibility comes from having a special parameter that controls nonlinearity – the order of the polynomial. And, as discussed in the previous section, one can use either model selection or model averaging methods to address this order parameter.

Another type of nonparametric regression is one where no parametric equation is written down at all. As we saw in Sections 2.4.3 and 2.4.4, it is possible to interpret the output of regression as a conditional mean of the response variables. So, if all we want from our statistical model is the ability to predict a y value from an x value, why can we not simply average the y values in the data, for a given x value, and call that average our prediction? Indeed, we can. The fancy name for that idea is kernel regression.

Of course, there are a few details which one has to worry about. For instance, suppose we have our data (x_i, y_i), and we want to make a prediction for a new case – call its x value x_0. If x_0 does appear in the data set, i.e., if it is one of the x_i, then the idea is to average all the y-values in the sample whose corresponding x is x_0. Then we can use that average value as our prediction for the y-value associated with x_0. But what if x_0 does not appear in the data? Then, the natural thing to do would be to find the nearest x value in the sample, and do the same thing as before. But that opens up a possibility: Why not average the y-values not just for a given x, but also for some nearby x's? The only other question is how near is near? Usually,

[9] Of course, one can turn all of this into p-values, which in turn can be used to reject (or not) the null hypothesis that two models are equivalent. But, in my opinion, the boxplots offer a more complete comparison of the two models.

[10] Technically, bagging refers to averaging the predictions of different models developed on different bootstrap samples, but model averaging refers to averaging the predictions of different models on a given data set.

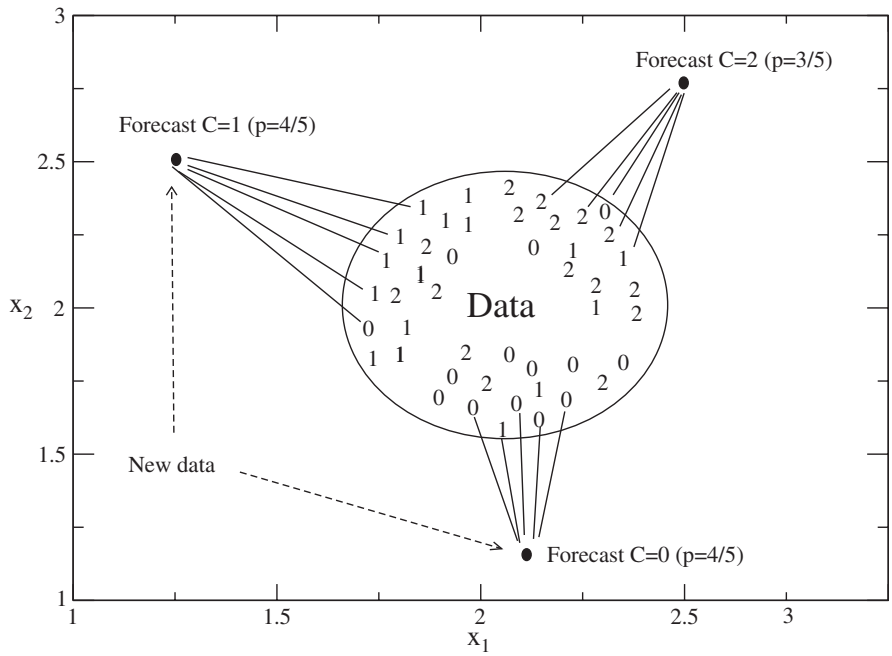


Fig. 2.5 An illustration of a K-Nearest-Neighbor model with K = 5

when a question like that comes up, one smart thing to do is to simply average over all the possible values, but simply give a heavier weight to the close ones. In other words, our prediction for the y-value corresponding to x_0 is the weighted average of all the y-values in the data, with the weights falling off with the distance between x and x_0. The weights are called kernels, and one very common one is a Gaussian $\sim \exp^{-\omega\,(x-x_0)^2}$. The parameter ω is called the smoothing parameter; it is the parameter that allows for the model to fit a wide range of relations. It is the analog of the order of a polynomial. It is the parameter that makes kernel regression nonparametric! For small values, the output of the model is practically the original data, i.e., the model overfits; For large values, the output is just the overall mean of the y values, i.e., the model underfits. For intermediate values, the output is the local average of the y values, i.e., the conditional mean of y, given x. The question, then, is what is an intermediate value? Again, the model selection methods described in the previous section are capable of providing the answer.

Note one other difference between polynomial regression and a kernel regression model: When asked to make a prediction for a new case, polynomial regression does not require the data that was employed for estimating the parameters in the polynomial – one simply uses the polynomial itself for making a prediction. But kernel regression requires the entire data set, because it needs to identify the cases in the data that are nearest to some case, and then average them in the way described above. As such, there is no training phase in kernel regression. For a particular application, this may be a pro or a con.

This same, almost trivial, idea of averaging over nearby points comes up in classification problems, under the name K-Nearest-Neighbor (KNN). To make a prediction of class-membership for a new case, KNN finds the K nearest cases in the sample data, identifies the "majority vote" among those K cases, and issues the corresponding class as its prediction. An example with K = 5 is given in Fig. 2.5.

The next extension of this simple idea is to associate a probability with each categorical prediction. The simplest way is to take the proportion of the majority class among the nearest neighbors, and treat that as a probability; these are the numbers shown as $p =$ in Fig. 2.5. However, these probabilities are not truly $p(C|x)$, which is what we would need for making predictions. But a model called Naive Bayes can be developed in conjunction with KNN to produce class-membership probabilities (Frank et al. 2003; Jiang et al. 2005). Needless to say, model selection or model averaging can be used to address K.

The main point of this section is to show that nonparametric models, in spite of their name, are not too different from their parametric counterparts. One still needs to address overfitting and model complexity of nonparametric models; and moreover, the notion of a distribution is always lurking around. As we will see in the next section, neural networks are no different in this respect.

2.8 Multilayered Perceptron

There have been numerous claims made about the stature of Multilayered Perceptrons (MLP) – or Neural Networks (NNs).[11] Most, however, are exaggerated or even blatantly wrong. In this section, I show that NNs are just another regression and classification tool, closely related to many of the classical models mentioned in the previous sections. NNs do, however, have certain properties that make them useful on a wide range of problems.

In fact, I think it is sufficient to mention just one of these properties, namely the manner in which neural nets handle the "curse of dimensionality". The best way to illustrate this point is by returning to polynomial regression. How many parameters does a polynomial regression have? First, recall that the number of parameters in a model is basically one of the factors that controls overfitting; too many parameters increase the chances of overfitting data. Also, too many parameters increase the uncertainty in each parameter's estimate. All of this makes perfect intuitive sense if we remind ourselves that all we have at our disposal is some finite sample, and that this sample contains some information that we are trying to distill into a model. The finite information in the data can only go so far in estimating the unknown parameters of the model.

Returning to the question of the number of parameters in a polynomial, if we have only one predictor, then a polynomial of order M will have $M + 1$ parameters. That is not too bad. But for p predictors, then we have many more because of the interaction terms, i.e., the cross terms between the predictors. In fact, the number of coefficients that must be estimated grows exponentially with p (Bishop 1996, p. 32). This makes it difficult to keep up with the demands of the model in terms of sample size. By contrast, as we will see below, the number of parameters in neural nets grows only linearly with the number of predictors. Meanwhile, they are sufficiently flexible to fit nonlinearities that arise in most problems. In short, they are "small" enough to not overfit as badly as some other models, but "big" enough to be able to learn (almost) any function.

Now, let us talk about the MLP. In terms of an equation it is simply a generalization of the regression equation $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_P x_P = \sum_{i=0}^{P} \beta_i \; x_i$, which we saw at the start of Section 2.6:

$$y(x, \omega, H) = g\left(\sum_{i=1}^{H} \omega_i \; f(\sum_{j=1}^{N_{in}} \omega_{ij} x_j - \theta_i) - \theta\right). \tag{2.31}$$

The θ and ω terms are the analogs of α, β in the regression equation. They are the parameters of the network – also called *weights*. The picture that accompanies equation (2.31) is shown in Fig. 2.6. This network would be referred to as an MLP (or NN) with N_{in} input nodes, one output node, and two hidden layer of weights (or equivalently, one hidden layer of nodes). The generalizations to multiple output nodes, or multiple hidden layers, are all straightforward.

Let us count the parameters: Each line connecting nodes in Fig. 2.6 is a weight, and there are $N_{in}H + HN_{out}$ of them. But, each of the hidden nodes and the outputs is also accompanied by a parameter – the θ's in equation (2.31) – and there are $H + N_{out}$ of them. In sum, the network has $(N_{in} + 1)H + (H + 1)N_{out}$ parameters. Note the linear growth with N_{in}. This is how neural nets manage to address the aforementioned curse of dimensionality. For a given number of predictors, they do not have too many parameters, at least compared with polynomial regression.

H is the number of hidden nodes, and it plays the role of the order of the polynomial regression. $f(x)$ and $g(x)$, called activation functions, are some prespecified functions; their job is to transform the predictors x_j, just like we did in Sections 2.5.1 and 2.5.3 when we squared x and then did regression between y and x^2. The only difference is that in neural nets circles, one does not use powers of x, but functions that look like a smoothed step function. Such step functions are generally called sigmoidal, and two

[11] Neural Networks, in general, come in a variety of types. Consult Masters (1993) or Bishop (1996) for a taxonomy. Usually, however, an in this chapter, neural network and multilayered perceptron are used interchangeably.

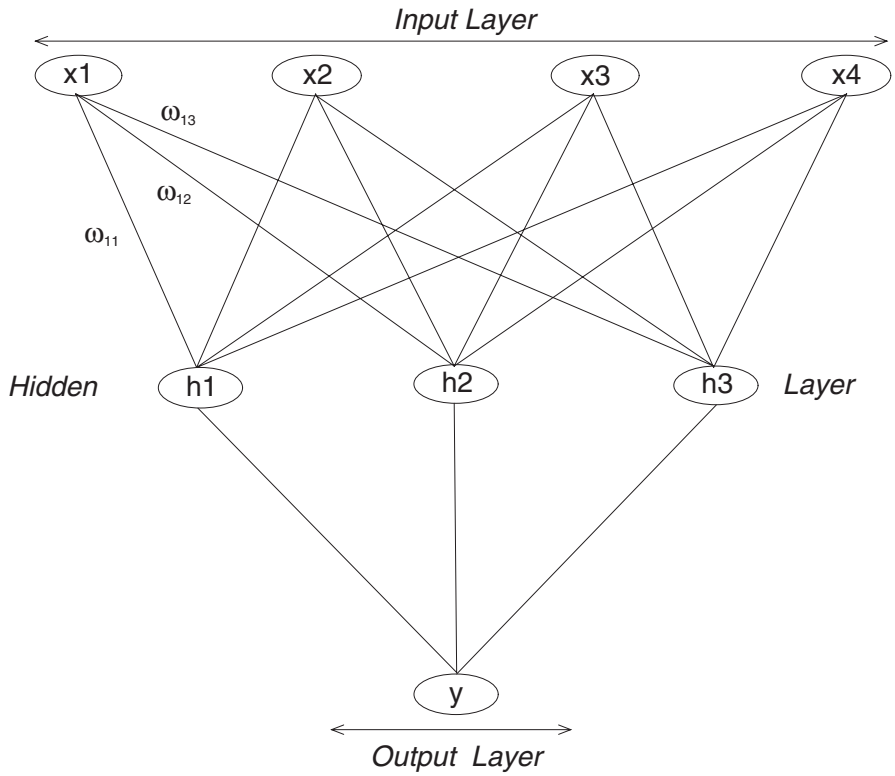


Fig. 2.6 An MLP with four input nodes, three hidden nodes on one hidden layer, and one output node. Some of the weights are also shown

common choices are $\tanh(x)$ and $1/(1+e^{-x})$, with the latter also known as the logistic function, whom we met in equation (2.27) when dealing with discriminant analysis. For regression problems, one usually uses one of these for $f(x)$, and then sets $g(x) = x$; this combination of f and g allows the network sufficient nonlinearity, while allowing the output of the network to take the full range of values that a continuous target variable usually takes. For classification tasks, ones sets both f and g to some sigmoidal function. These are all common but not universal or absolutely unique choices.

Without going into details, we can now see why an MLP can fit (almost) any data. Just think of $f(x)$ as a basis function; equation (2.31) then says that any function relating y to the x_i's can be approximated with the function given in equation (2.31), given some values for the parameters, and given a sufficiently large H. For those readers who feel comfortable with Fourier decomposition, this is analogous to Fourier's claim that any time series can be written as a sum of a bunch of sines and cosines. The MLP analog is that any function can be written as a linear combination of a bunch of sigmoidal functions.

Incidentally, I say "any function", but I should be saying "most functions." There are some functions that cannot be represented by equation (2.31). Many of them can be represented by MLPs with more hidden layers. One can continue making equation (2.31) increasingly flexible in order to allow it to fit increasingly complicated functions, but in most practical problems it is unnecessary to go beyond one hidden layer of nodes.

2.8.1 Estimating Weights – Training

Let us recall where we are going. The task is to take some data set that relates some number of predictors x_i to a response y, and then estimate all the parameters appearing in equation (2.31). When we are done, we can take our MLP and start predicting with it; feed it inputs, and see what it predicts for the response.

However, as you can probably guess from our regression experience, there are two types of parameters in equation (2.31), and they require separate treatment. The estimation of the weights is called training, and H can be addressed within either a model selection or a model averaging scheme.

Again, just as in regression, training involves minimizing some error function, denoted E. But keep in mind that the minimization is just a pragmatic approach to the maximization of the probability of obtaining the data.[12] For regression problems the common choice is the mean squared error (MSE)

$$E(\omega) = \frac{1}{n}\Sigma[y(x,\omega) - t]^2, \tag{2.32}$$

and for classification problems, it is the so-called cross-entropy,

$$E(\omega) = -\frac{1}{n}\sum[t \log y(x,\omega) + (1-t) \log(1 - y(x,\omega))], \tag{2.33}$$

both of which are related to the likelihood of the data, mentioned in Section 2.4. One looks at the difference between the truth and the prediction, while the other looks at (the log of) their ratio.

In these so-called error functions, ω refers to all the parameters in equation (2.31), except H. The sums are over the n cases in the training data, but for the sake of clarity, I have suppressed the corresponding index. So, $y(x,\omega)$ is given by equation (2.31), and the t are the corresponding targets. Note the slight change in notation: in statistical circles, the observed data are denoted (x_i, y_i), but in AI circles, they are written as (x_i, t_i) with $y(x)$ reserved for the output of the model. This explains the difference between equations (2.32) and (2.4).

For regression problems, t takes a continuous range, but for classification problems, it is a discrete variable. For a two-class problem, it is sufficient to let t be 0 or 1, denoting the two classes. But if the problem has more classes – especially if the classes have no inherent order – then it is wise to encode them among several output nodes. For example, if the problem has three classes, then our MLP should have three output nodes, taking the values (1,0,0), (0,1,0), (0,0,1) for the three classes, respectively. Note that these are the target variables for training a classifier; when the classifier is being employed for prediction, then the outputs are not necessarily 0 or 1. In fact, they are continuous numbers between 0 and 1 (because $g(x)$ is logistic), and it can be shown that the output of the jth output node is the conditional probability of belonging to the jth class, given the inputs (see below). This scheme is called 1-of-c encoding.

Although the choice of mean squared error in equation (2.32) is not surprising, the expression appearing in equation (2.33) may seem arbitrary. However, the fact is that both of these expressions are derived from assumptions on the underlying distributions. Just as in Sections 2.4.3 and 2.4.4, the minimization of equation (2.32) is equivalent to the maximization of the probability of data, if the errors are normally distributed. Similarly, for a classification problem, comparing the logarithm of equation (2.11) with equation (2.33) reveals that the binomial distribution has been assumed at some stage. As such, one should be cautious of claims that MLPs are assumption-free, at least if one desires a probabilistic interpretation of the outputs.

More specifically, recall that the minimization of mean squared error assures that the output of a regression model can be interpreted as the conditional mean of the target values, given the input (Sections 2.4.3 and 2.4.4). The same is true of an MLP in regression-mode, i.e., when the targets are continuous and we minimize mean squared error (Bishop 1996). The analogous result for a classification MLP is this: if the activation functions f and g are logistic, then the minimization of cross-entropy guarantees that the output is a posterior probability of class membership, given the inputs (Richard and Lippmann 1991). In equation form: $y(x,\omega) = p(C|x)$. This probability is precisely the type of probability required for assessing or conveying uncertainty in predicting class membership. By the way, for the output to have this probabilistic interpretation, the various classes in the training data must appear in their "natural" (or climatological) proportion. If the number of cases in each class is artificially equalized – as is sometimes done – then the output must be scaled in a special way to maintain the probabilistic interpretation (Bishop 1996).

Now, let us return to the minimization task. Given the nonlinearity of $E(\omega)$ on ω, it is impossible to

[12] In the Bayesian approach, one actually maximizes the "other" probability, i.e., the probability of obtaining some parameter, given the data (MacKay 1996; Bishop 1996).

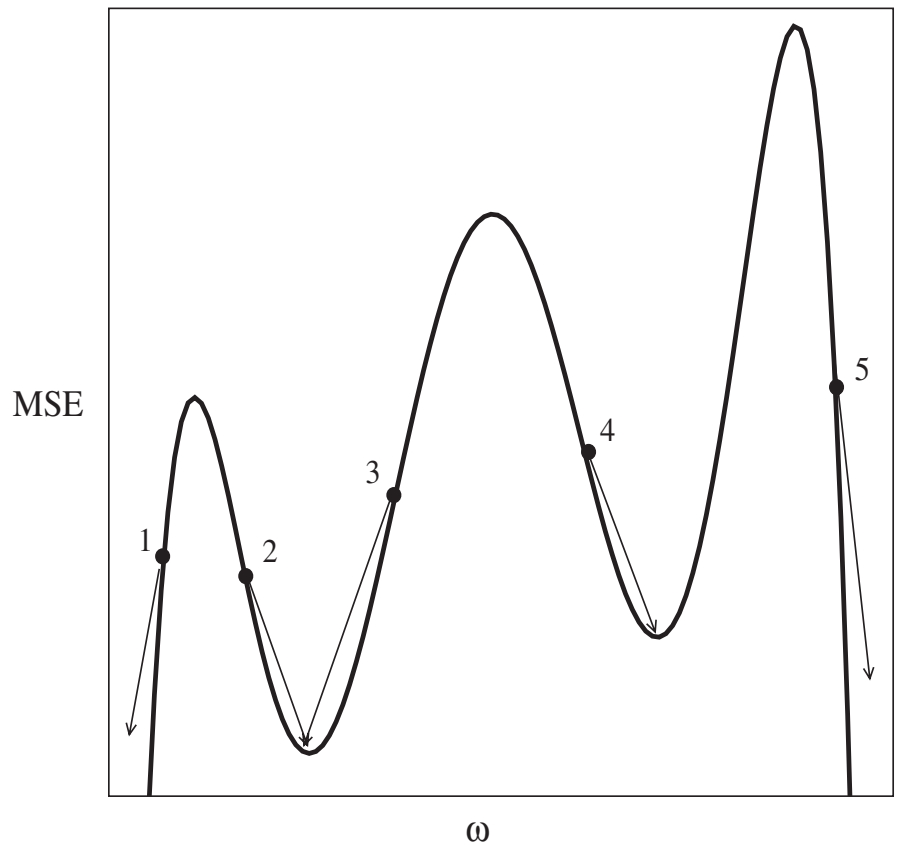


Fig. 2.7 An illustration of local minima

solve the $\partial E/\partial\omega = 0$ equations exactly. The usual approach is to adopt some iterative approach that converges to the minimum of $E(\omega)$. The brute force approach could be something like this: Vary all the weights over their full range in some small increments. Calculate the E at each increment, and then select the weight values that yield the lowest E. The good news is that there exist much smarter ways of finding the minimum of E. Some of the more popular ones are gradient descent, backpropagation, (scaled) conjugate gradient, simulated annealing, genetic algorithms, etc. (Masters 1993; Bishop 1996; Haupt and Haupt 1998). They all involve initializing the weights randomly, and then proceeding to vary them in increments and in a way that is designed to minimize the error function.

The fly in the ointment is that there almost always exist (too) many local minima. In other words, if you pick some random initial weights and start training (i.e., minimizing E), you will find that after some time E does not seem to decrease any further. You might think that you have found a minimum. However, if you start training from different random initial weights, you will probably find a different minimum – different both in the value of E and the corresponding final weights. These are all local minima.

Figure 2.7 provides an qualitative illustration of local minima. The x-axis represents a single weight, while the y-axis denotes the error function being minimized, say, MSE. If the initial weight corresponds to the points labeled 2 or 3, then a training algorithm will reduce the MSE, and land us in the global minimum. An initial weight corresponding to point 4 will land us in a local minimum. In this illustration, the local minimum is actually not too bad, because the corresponding MSE is close to the one at the global minimum. As such, even this local minimum would suffice. By contrast, if the network's initial weights place it at points 1 or 5, then the weight shrinks continually, or diverge to infinity, respectively. What awaits in these two extremes is either a deeper global minimum, or nonphysical solutions. One never knows

with certainty – hence, the *art* of training a neural network.

There are many methods for both avoiding local minima and escaping them, but none of them guarantee that a global minimum has been found, at least not in practice. Actually, it turns out that the desire to find *the* global minimum is unnecessarily obsessive. In statistical settings, i.e., dealing with data, a sufficiently deep local minimum usually suffices. What is "sufficiently deep"? The answer depends on the specific problem/data. So, it is best if one just trains from many (say 100) different initial weights, and plots the distribution of the final E's. This will show not only the most likely local minima but also the "chance" of landing in a global minimum. The code provided here does allow one to explore the various local minima, and I will illustrate their effect in Section 2.8.3.

2.8.2 Estimating Complexity – Model Selection and/or Averaging

I have said that H is the analog of the order of a polynomial, and so, it affects the nonlinearity of the relationship between the input and output variables.[13] The fact is that there are several parameters which effect the nonlinearity of a network. In addition to H, there is the magnitude of the weights, and the number of iterations toward the minimum (also called epochs). In the context of MLPs, then, model selection refers to finding the "optimal" values for these parameters. Although, it turns out that we do not really need to optimize every one of these parameters, it is good to know that they exist.

To see how the magnitude of the weights enters the game, examine the logistic activation function, and its Taylor series expansion: $f(x) = \frac{1}{1+e^{-\omega x}} = \frac{1}{2} + \frac{1}{4}(\omega x) - \frac{1}{8}(\omega x)^3 + \ldots$. Suppose we are dealing with a regression problem, and so we pick the identity function for g in equation (2.31). If the magnitude of ω is "small", then $f(x)$ is linear in x, and the whole MLP equation (2.31) becomes linear in x. In other words, if the weights of the MLP are small, then it is a linear function, regardless of how many hidden nodes it has, or how many epochs it is trained for. Said differently, the entire MLP is nothing but a linear regression model. At the same time, for large $|\omega|$, $f(x)$ is very nonlinear in x. Then, even for a moderately small number of hidden nodes, the MLP can end up being quite nonlinear.

The magnitude of the weights itself is effected by two other "knobs" that enter at the training phase: the range of the initial weights, and what the training algorithm does to them. The first one is controlled through the standard deviation (or range) of the distribution from which the random weights are drawn, and the second one is controlled by the introduction of a term in the error function being minimized, e.g., equations (2.32) or (2.33). It is called the *weight decay* term, and its job is to penalize the error function when the weights get too large. In its simplest implementation, it looks like $\nu \sum \omega^2$, where the sum is over all the weights in the network. In this form, the single coefficient ν determines the over-all effect of weight-decay. The larger ν is, the smaller the weights, and the more linear the network becomes.

Looking at the Taylor series expansion of $f(x)$, it is clear that the adjectives "small" and "large" are relative to the size of x; and that is where another parameter enters the scene, namely the range of the inputs. If the inputs vary over a small range, then even a large ω may not yield a large enough (ωx) to make the logistic function nonlinear. To control this effect on nonlinearity, a common practice is to either scale the inputs to some reasonable range (e.g., -1 to $+1$), or to standardize them as $z = (x - \overline{x})/s$, where x refers to all the cases in one input, and $\overline{x}$ and s are the sample mean and standard deviation of that input. This transformation of the inputs is statistically pleasing because the transformed variables (also called z-scores) will have a mean of zero and a standard deviation of 1, placing different inputs all on the same footing. Anyway, the point is that we also need to worry about the range of the inputs, and preprocess them in some intelligent way.

Finally, to control the number of epochs, i.e., the number of times the weights are updated on their way to the minimum, one does what is called called early stopping. And that is exactly what it sounds like: one

[13] $H = 0$ means an MLP with no hidden layer of nodes. Such a model, also called a Perceptron, is equivalent to linear or logistic regression depending on the choice of the activation function. Marzban (2004) shows the close relationship between logistic regression, discriminant analysis, and MLPs.

simply terminates training. When to stop is determined by monitoring the performance of the network on a validation set. Whereas the training error continues to decrease, the validation error decreases initially, and then either fails to decrease or even begins to increase. One simply halts the training when the validation error begins to rise. This hold-out method (Section 2.6), however, has some problems with small data sets (Goutte 1997).

The aforementioned parameters are not all independent. For example, if H is such that the parametric structure of equation (2.31) simply does not allow a proper representation of the underlying function relating the inputs to the outputs, then a larger number of epochs will not get us anywhere. As such, it is recommended to fix most of these parameters, and employ model selection methods (Section 2.6) to address the rest. One can set H to a large value (even exceedingly large), initialize the weights from a wide distribution, and then let ν control nonlinearity. Alternatively, one can set $\nu = 0$, and then vary H. In both cases, one should set the number of epochs to some very large number. In this chapter, I will follow the latter. Having said all of that, I must also confess that there is an element of art in deciding what is large and what is small. The best way is to play and get a feeling of how things go.

And there is more! One facet of neural net development which I have ignored is their training method. Backpropagation, conjugate gradient, Levenberg-Marquardt, simulated annealing, and genetic algorithms, are all training algorithms designed to minimize some error function. They do their business differently, though. The first three are based on the derivative of the error function with respect to the parameters; simulated annealing relies on ideas from condensed matter physics to minimize the error function without taking any derivatives, and genetic algorithms (Chapter 5) do the same but with ideas from evolutionary biology. The trouble is that they too come along with a host of parameters the user is expected to set. A good discussion of all of these methods is given in Masters (1993).

To close this section, let us recall again, that the reason we are worrying so much about overfitting is that we desire to develop a model that performs well in the population, and not specifically on any given sample. I will now turn to illustrations of all these ideas.

2.8.3 A Classification Example

Model selection methods are designed to identify the best model, or the true model (if one does exist). By "best" we mean good predictions on unseen data. So, let us test this claim. Let us pick a known function, create some data from it, and see if our model selection method can identify the underlying function from the data alone. We already considered such an exercise in Section 2.6, with polynomial regression. So, this time, let us try a classification problem attacked with neural nets.

To that end, I picked an MLP with two input nodes, H = 6 hidden nodes, and one output node. I then generated 1,000 random pairs of numbers between -1 and +1, and fed these as inputs to the network. With randomly initialized weights, I added a bit of noise to the outputs (normal, with a mean of zero and a standard deviation of 0.3), and assigned 0 (1) to the target, if the output was less than (greater than) 0.5. The resulting data are shown in Fig. 2.8. The boundary between the 0s and 1s in Fig. 2.8 is then our true underlying boundary. Note that I did not use an MLP to learn any data; I used an MLP structure to *generate* data. The question is if our model selection method will be able to identify the correct underlying boundary, i.e., the correct number of hidden nodes, i.e., 6?[14]

So, I took the generated data – all 1,000 of them – and went to work. Given that we are dealing with a classification problem, all the activation functions were logistic; the error function was cross-entropy without a weight-decay term, and I allowed the training algorithm to converge to minimum (i.e., no early stopping). I then developed three networks with $H = 0, 4$, and 6 hidden nodes. Why these specific values? You will see! The model selection method I used was the double-bootstrap, as described in Section 2.6. The number of "outer" bootstrap trials was 60; in other words, I took 60 different samples from the data, each of size 700. Then, for each of these 60 samples, I used proper bootstrap to get a point-estimate of the prediction error. The number of "inner" bootstrap trials

[14] Of course, there is more to the boundary in Fig. 2.8 than just the number of hidden nodes of the network; namely, the specific values of the weights of the network. But, as I have mentioned, upon training a network with six hidden nodes on the training set, the only ambiguity in the particular weights is due to local minima. As such, we should still try a few other initializations and get a sense of the range of variability.

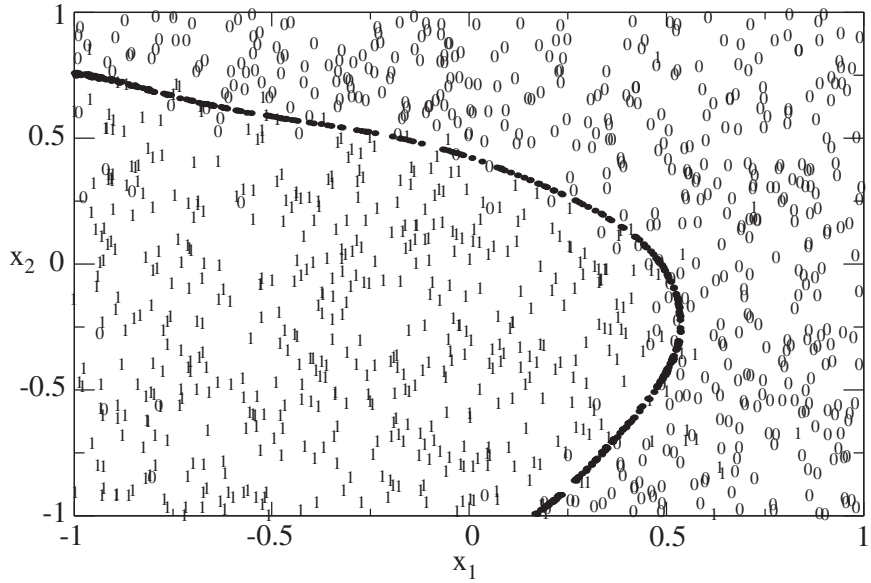


Fig. 2.8 Artificial data with two predictors (x_1, x_2), and one binary target (0/1) made with an MLP with six hidden nodes

was 70. Recall, that the "outer" boostrap gives us the sampling distribution of the prediction error. This way, we will have some sense of the uncertainty of the prediction errors for each of the three models. These numbers were chosen almost randomly; however, there are some theoretical reasons to take the number of outer trials to be approximately the square root of the inner trials (Kuk 1989).

Figure 2.9 shows the boxplots for the average (over "inner" bootstrap trials) training errors (left) and prediction errors (right). As expected, the training error falls off with increasing complexity. However, as seen in the right panel, the prediction errors are generally lower for the model with four hidden nodes. The $H = 0$ model underfits the data, and the $H = 6$ model overfits it. Yes, the $H = 6$ network overfits the data, and so, it is not the best model for the data at hand. So, are we to conclude that our model selection method failed, because it did not arrive at the correct answer, 6? No, the method did not fail. The resolution to the discrepancy is in the way I generated the data. Revisit the beginning of this section, where I described the way the data was generated. I did indeed take a network with $H = 6$ hidden nodes, but such a network includes, as special cases, functions that can be represented with lower values of H. Just imagine setting some of the weights in the $H = 6$ network to zero, so that it would represent only functions learnable by $H = 4$, or even $H = 0$ networks. It is doable, and in this case it happened. In other words, when I assigned random weights to the $H = 6$ network, the resulting network was one that could easily fall within the space of $H = 4$ networks. That is why the model selection method arrived at the $H = 4$ model as the best one for the data at hand.

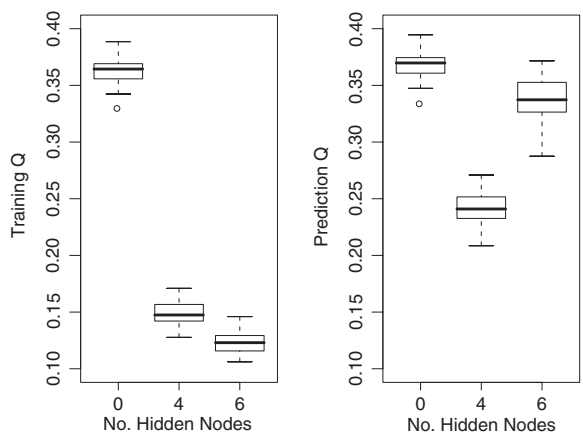


Fig. 2.9 The performance of the various models in terms of the average training Q (left) and the Prediction Q (right), on 60 "outer" bootstrap trials. A brief description of boxplots is in order: The dark horizontal line in the middle of each box marks the median, and the lower (upper) edge of the box marks the first (third) quartile of the data. The whiskers at the ends of the vertical lines denote the range of the data, but only after the outliers (the circles) have been removed. There exist different notions of an outlier, but here everything beyond 1.5 × (third quartile – first quartile) is defined as an outlier.

That is one explanation of the apparent failure of the methodology. The other explanation is this: The $H = 4$ model is the right one *for the data at hand*, i.e., the 700 cases that are selected for actual training. Had our original data been larger, it is possible that a larger value of H would turn out to be optimal. It is important to realize that the notion of "the best" model is contingent on the available data. And by the way, this is why I chose not to consider networks with $H > 6$ – because all larger networks would overfit even more so than the $H = 6$ network. As for the specific choice of 0, 4, and 6: there is no particular reason! You may choose 0, 3, and 6 if symmetry is pleasing; I suspect the prediction errors will be comparable to the 0, 4, 6 cases. I usually take some even numbers, but I try to include 0, because a $H = 0$ network is equivalent to logistic regression – a simple but powerful model that makes for a good benchmark.

Now, let us return to Fig. 2.9 and discuss the boxplots a bit more. Yes, the $H = 4$ model is producing generally lower prediction errors than the other two models; but what about the spread in these errors, reflected in the range of the box as well as the whiskers? In this case, we are lucky in that there is almost no overlap between neighboring models. So, modulo the assumption that the prediction errors are not paired between the models (see below), we may conclude that the model with $H = 4$ hidden nodes is not only the best model, but it is better than the other two in a statistically significant sense. However, if there had been significant overlap between the boxplots, then we would have to conclude that, given the data, we cannot decide which model is better. Said differently, we would have to conclude that all the models are statistically equivalent. That is why it is important to look at these boxplots.

It is worthwhile to view the above results in one other way. The top three panels in Fig. 2.10 show the histograms of the average training errors (on left) and the prediction errors (on right), for the three models with $H = 0, 4$, and 6. Again, note the steady shift of the training histogram to lower errors as the model becomes more complex. From the prediction histogram, it can be seen that the model with four hidden nodes has lower errors than the $H = 0$ model and the $H = 6$ network.

Let us remind ourselves what these histograms actually tell us. In addition to selecting the right model, they also tell us about the error the model is expected to make in the long run. For example, the histogram for the $H = 6$ network tells us that if we use this network for prediction, then we expect to make an error of about 0.24 ± 0.05, these numbers being the mode and the standard deviation of the prediction histogram. Granted, this error is in cross entropy – not exactly a commonplace measure. But all of these histograms could have been for some other measure, like root-mean-square error (RMSE) of surface temperature, in Centigrade. Suppose the mode and the standard deviation of the prediction histogram of temperature RMSE turned out to be 10 and 1.2, respectively. Then, we would conclude that our model is expected to make an error of $10°C \pm 1.2°C$ when it is used on new data. This is the kind of important information that should accompany any predictive algorithm. Without it, we have no idea how good the algorithm really is.

And there is still one more way of viewing the results. The bottom panel in Fig. 2.10 shows what I call the "tv-diagram" (Marzban 2000). It is a scatterplot of all "outer" bootstrap trials, with the average training error on the x-axis and the prediction error on the y-axis. The cluster of 60 dots to the right corresponds to a network with zero hidden nodes, the middle/low cluster is for an $H = 4$ network, and the cluster immediately to the left comes from a network with $H = 6$. The scatter of the dots gives one a visual sense of the spread in cross-entropy, as well as the correlation between training and prediction errors. Again, it can be seen that the training errors for models with more hidden nodes are systematically lower, but the $H = 4$ model has generally lower prediction errors than the other two. Furthermore, the training and prediction errors for $H = 0$ network are highly correlated; i.e., a lower training error almost guarantees a lower prediction error. This, however, is not true of the larger networks, at least not to the same extent. There appears to be a weakening of the correlation for larger networks, and I suspect it can be attributed to the increase in number of local minima as a network grows larger. More on local minima, below.

With Figs. 2.9 and 2.10, we can feel relatively confident that the correct model for the data at hand is a network with $H = 4$. As discussed in Section 2.6, at this stage we would train such a network on the entire data, and ship it out for use. Alternatively, we could ship out all 60 networks with $H = 4$, and average their predictions.

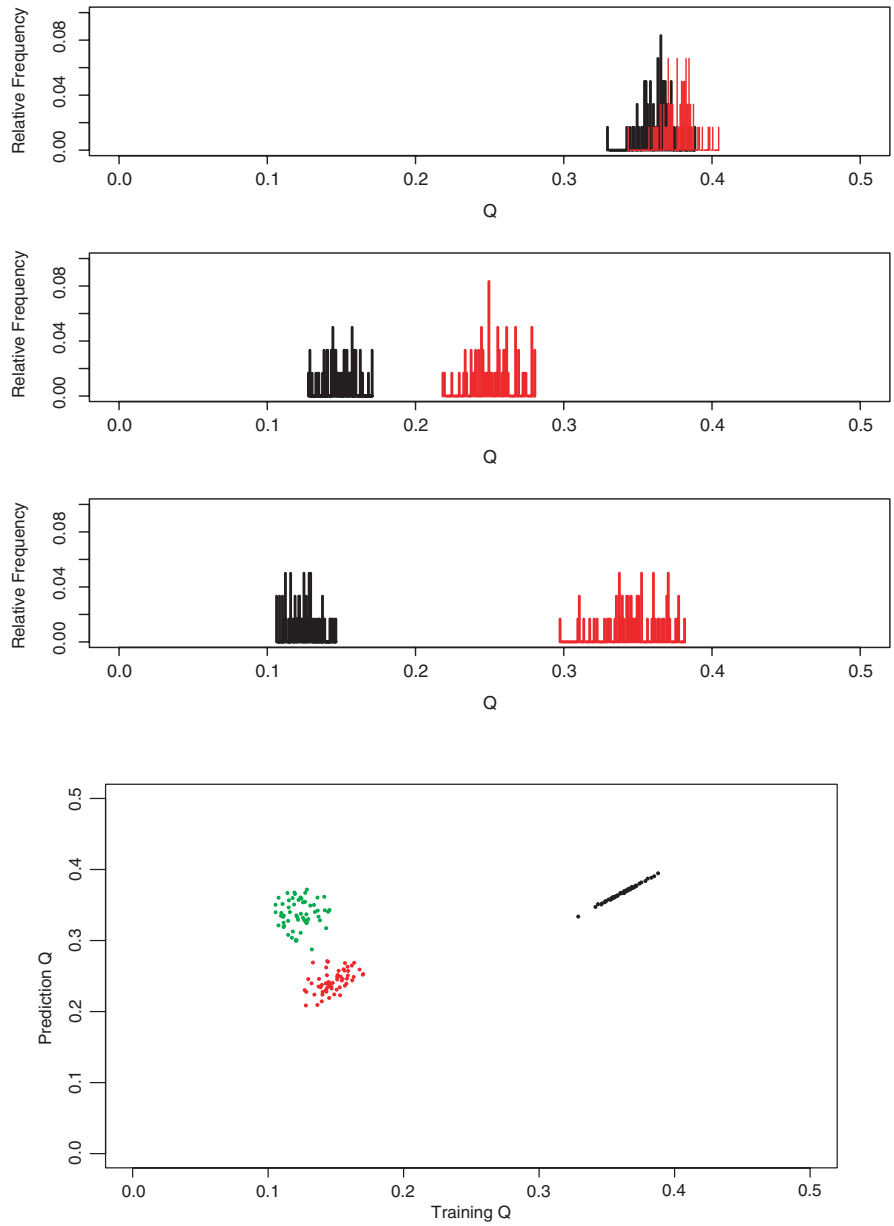


Fig. 2.10 The top three panels show the histograms of the prediction error, Q, for $H = 0, 4, 6$, respectively. The bottom panel shows the scatterplot of average training errors and prediction errors ("tv-diagram") for 60 "outer" bootstrap trials for MLPs with $H = 0$ (most left cluster), 4 (middle, and lowest cluster), and 6 (most right cluster) hidden nodes

Now, what about the assumption (from five paragraphs above) that the prediction errors are not paired between the two models being compared? The notion of "paired data" was discussed in the penultimate paragraph of Section 2.6.1. There, I pointed out that a comparison of two sets of numbers, in terms of the boxplots of each set, is valid only if the two sets are independent of each other. But, here in comparing the set of prediction errors from one model to that of another model, the two sets of prediction errors are not independent, because they are estimated from the same outer bootstrap trial sets. The solution is to simply take the difference, across models, of all 60 prediction errors, and then look at the boxplot of the differences. Figure 2.11 shows these boxplots. "4-0" denotes the difference "(prediction error of $H = 4$ network – prediction error of $H = 0$ network)". Similarly, "6-4" denotes (prediction error of $H = 6$ network − prediction error of $H = 4$ network), etc. So, given the all-negative values of the "4-0" boxplot, we

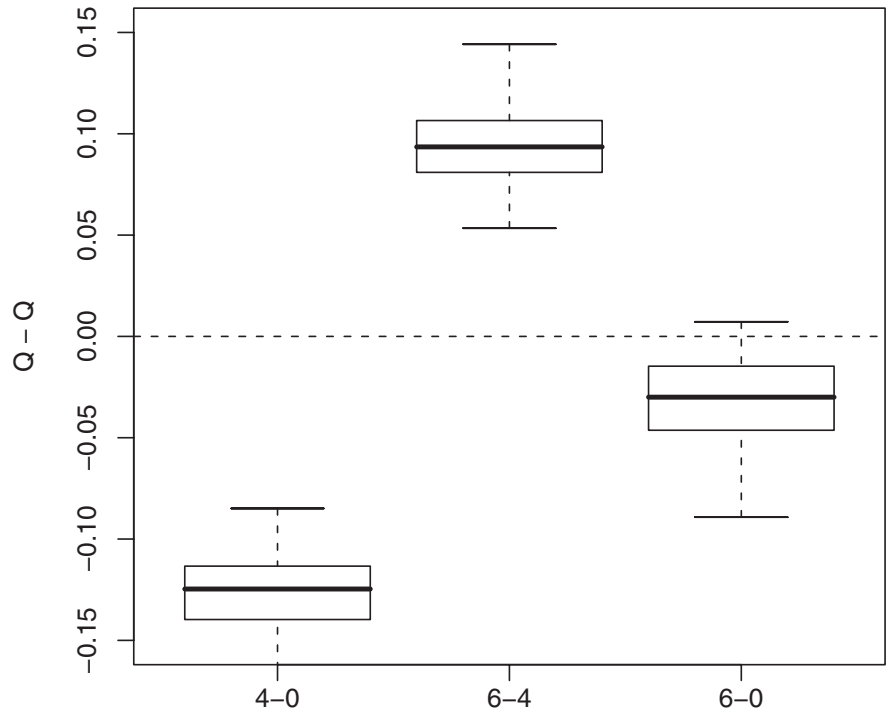


Fig. 2.11 Boxplots of the difference of prediction errors. "4-0" denotes the difference between the prediction errors of a $H = 4$ network and that of an $H = 0$ network. Similarly for "6-4" and "6-0"

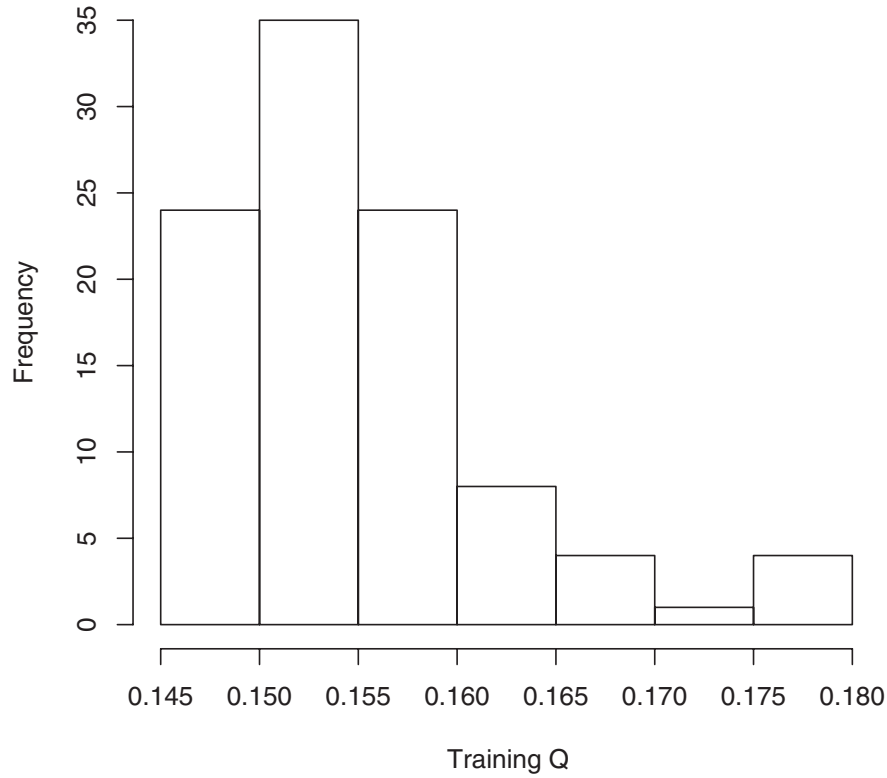


Fig. 2.12 Histogram of the local minima

conclude that the prediction errors of the $H = 4$ network across the outer bootstrap trials are all lower those of the $H = 0$ network. Similarly, the "6-4" boxplot implies that for each and every outer bootstrap trial, the $H = 6$ network produces larger prediction errors than the $H = 4$ network. The last boxplot suggests that the $H = 6$ network and the $H = 0$ network are statistically indistinguishable. In short, the conclusion is still the same – that the best model is the network with four hidden nodes.

Finally, it is possible to "explain" the spread in the prediction errors. In MLPs, there are at least two sources of variation: due to bootstrap resampling and due to local minima. To assess how much of the variation in the error values is due to local minima, I selected a single bootstrap trial and trained a $H = 4$ network 100 times starting with different initial weights. Figure 2.12 shows the histogram of the training errors. Comparing the range of the values on the x-axis in this figure with those appearing in Figs. 2.9 or 2.10, one can conclude that a good deal of the variation in the errors is due to local minima. Marzban (1997) attempts to compare the magnitude of these two sources of variation. But as far as I know, no general results exist on this matter.

Although we have expended a good deal of energy to arrive at the "best model", it is important to reiterate that often a range of H values will produce comparable results, all of them equally good. This is analogous to the case of local minima, where it is sufficient to find a relatively deep local minimum, and not necessarily the absolute global minimum. By model selection, we just want to assure that H is in the right ballpark.

2.9 Final Comments

Preprocessing of the data is a crucial part of any model building task. In the examples employed in this chapter, I was unable to make this point, because the data were artificially made. In practice, a great deal must be done to both the inputs and the outputs, in order to make the data conform to requirements of the model, either mathematical constraints or probabilistic. Either way, one should make the job of the MLP as easy as possible. Masters (1993) offers a good deal of guidance, and Marzban (2004) does more of the same. And the reader can get a sense of different types of preprocessing by reading Chapters 12 and 13 in this book.

Some of the preprocessing involves just "looking" at the data. Scatterplots and histograms are the basic tools for that purpose. The utility of these two tools is even more impressive when they are conditional, i.e., conditioned on some other quantity. Figure 2.10 contains examples of class-conditional histograms (top three panels), and a scatterplot conditioned on H. Of course, these figures pertain to the outputs of the model, but similar plots can be viewed to study the data before they are fed to a model.

Another preprocessing step involves reducing the number of inputs as much as possible. This reduces the number of parameters in the model, and therefore reduces the chance of overfitting. One method relies on finding linear combinations of the inputs that contain most of the information in the data. In other words, one does data compression on the inputs. The classic method for this is called principal component analysis. Given data on p predictors, it finds p different linear combinations of the predictors (called principal components), and orders them in terms of the percentage of the variance in the whole data explained by each linear combination. Usually, the first few linear combinations account for most of the variance in the data.[15] So, instead of the p predictors, one can use only the first few principal components of the data as inputs into the model. I have included a piece of code that can do this, within a neural net code. The only draw-back of this practice is that it can actually lead to some loss of information, because principal component analysis does not refer to the target values at all; it is done on the inputs only. Bishop (1996) presents some pathological examples to illustrate this point. But in my experience, the examples are truly pathological in the sense that I have never come across one, in practice.

Finally, the performance of a statistical model can be assessed in terms of measures that are often different from the error function being minimized. The performance of a classifier is often assessed in terms of a hit rate and a false alarm rate, even though it is

[15] Chapter 8 discusses a nonlinear principal components analysis, where nonlinear combinations of variables are taken. Ironically, an MLP is employed to take the nonlinear combinations.

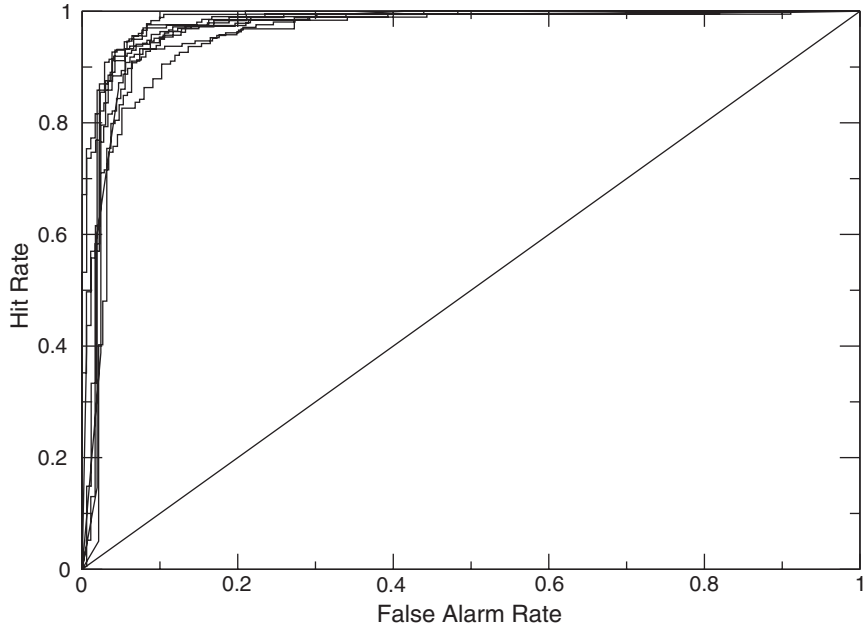


Fig. 2.13 ROC curves for 10 bootstrap trials

developed by minimizing cross entropy.[16] Many performance measures are discussed in Chapter 3, and the model selection methods described in Section 2.6 here allow for assessing their uncertainty as well. For example, the hit rates and false alarm rates from a classifier are often reported in the form of a Relative (or Receiver) Operating Characteristic (ROC) curve, which is a plot of the former versus the latter, for different values of a threshold placed on the output of the model; see Chapter 3 for a more complete description. Figure 2.13 shows 10 ROC curves from 10 bootstrap trials of the $H = 6$ network examined in the previous section. Such a plot clearly shows the range of variability of ROC curves. And this can help in model selection, if one desires to select a model with the best performance on ROC rather than on cross entropy. Do recall, however, that the above-mentioned issue of "paired data" affects how one should compare ROC curves, as well.

One last comment: Throughout this chapter I have presented the case that many (if not all) techniques generally considered as "traditional statistics" are really not that different from machine learning or artificial intelligence techniques. Prejudice may have swayed me to write in favor of the former; however, there is one feature of AI techniques which I believe has been unjustly used to their disadvantage. Specifically, some people have been referring to neural networks as a "Black Box", because one is generally unable to decipher what it is that they do. Those people have been using this fact – and yes, it is a fact – to argue against neural networks. There is, however, a fallacy in their argument: The opaqueness of neural networks is not a property of neural networks; it is a property of *the data* which the network is attempting to model. For example, suppose we have gone through our model selection method and have arrived at a neural network with multiple hidden nodes, and even multiple hidden layers, as the optimal model. As such, the underlying function relating the inputs to the outputs is probably highly complex, with strong nonlinearity and interactions among the variables. Well, that knowledge alone, quite independently of the neural network, is sufficient to render the problem opaque, because it is then generally impossible to explain the underlying function in a simple, meaningful way. For example, one can no longer uniquely quantify the predictive strength of a given input, because that quantity would depend on the values of all the other predictors. In short, if one knows from some source, e.g., from a neural network, that the relation underlying data is complex, then one

[16] Marzban and Haupt (2005) propose a method of directly minimizing the performance measure of interest, even when the performance measure is not a differentiable function of the model parameters.

should not expect to be able to explain the relationship in a simple way. Then, it is the problem itself which is opaque, and the neural network is simply the messenger who ought not be shunned.

To close, recall that we began this chapter with some elementary concepts in statistics, and have now naturally ended-up with what many might call "new" methods, e.g., neural nets. One of the aims of this chapter was to makethe transition sufficiently smooth in order to make it difficult to disambiguate between the new and the old. Another aim was to illustrate how basic statistical concepts (e.g., distributions, variation, etc.) are indispensable for a proper development of any model, now and old alike. Which brings us to the main theme of the book itself, namely the "red thread" that smoothly connects apparently disparate set of techniques and methodologies in environmental sciences.

2.10 Computer Code

```
#######################################################################
# This code performs classification with Linear Discriminant Analysis (LDA) or
# with a Multilayered Perceptron (MLP). It also has the option of using
# Principal Components (PC) as inputs; but it will require some revision by hand.
# It generates specific plots in Figure 2.8. The data consists of 2
# predictors and 1 binary target. The first few lines of the data, in a
# file called "class.dat" (not included in this book), look like:
#
# 0.71390560  0.17791940 0
# 0.38786700  −0.25662960 1
# 0.64317600  −0.73170410 0
# −0.61998590  −0.09877737 1
# −0.68146780  −0.36785700 1
#
# The first pair of numbers on each line are the values of the predictors,
# and the 3rd number is the target.
#######################################################################

rm(list=ls(all=TRUE))                          # Clears things up.
library(MASS)                                  # For LDA.
library(nnet)                                  # For MLP.
library(verification)                          # For ROC plot.

n.bootin = 70                                  # No. of inner bootstrap trials.
n.bootout = 60                                 # No. of outer bootstrap trials.
n.in = 2                                       # No. of input nodes.
n.hd = 6                                       # No. of hdn nodes, on 1 hdn layer.
n.out = 1                                      # No. of output nodes.
wtdecay = 0.00                                 # 0.00 for no weight decay.
range = 10.0                                   # Range of initial weights in MLP.
max.iter = 400                                 # No. of epochs.
eps = 1e-10                                    # To avoid log(0).

dat = as.matrix(read.table("class.dat", header=F))
n = dim(dat)[1]
```

```
n.trn = 700                                          # Size of samples on which bootstrap is done.
tpred = matrix(nrow=n.bootin,ncol=n.trn)             # Allocate memory.
vpred = matrix(nrow=n.bootin,ncol=n.trn)
err.trn = numeric(n.bootout)
err = numeric(n.bootout)

for(trialout in 1:n.bootout){                        # Start of sample selection.
set.seed(trialout)                                   # Set seed for reproducibility of results.
sampout = c(sample(1:n,n.trn,replace=F))             # Sample on which bootstrap is done.
input = dat[sampout,1:2]                             # Select inputs,
target = dat[sampout,3]                              # and target.
input = matrix(as.vector(input),ncol=n.in)           # Re-label the row numbers.
cases = seq(1,n.trn,1)                               # The index of the cases in the data.
e0 = matrix(nrow=n.trn,ncol=n.bootin)                # Allocate memory
e1 = matrix(nrow=n.trn,ncol=n.bootin)                # for storing errors.
e0[,] = e1[,] = NaN                                  # Makes it easier to compute E1, below.

for(trialin in 1:n.bootin){                          # Start of bootstrap trials.
set.seed(trialin)                                    # Set seed for reproducibility of results.
samp - c(sample(cases,n.trn,replace=T))              # Bootstrap = sampling with replacement.
# mod = lda(input[samp,1:n.in], target[samp] )       # For linear discrim.
mod = nnet(input[samp,], target[samp],
     size = n.hd, rang = range, decay = wtdecay,
     maxit = max.iter,entropy=T,linout=F)
for(i in 1:length(cases[samp]) )                     # Store the trn and vld predictions.
tpred[trialin,cases[samp][i]] =
         t(predict(mod,newdata=input[samp,]))[i]
for(i in 1:length(cases[-samp]) )
vpred[trialin,cases[-samp][i]] =
         t(predict(mod,newdata=input[-samp,]))[i]

for(i in cases){                                     # Compute trn and vld errors.
e0[i,trialin] = -(target[i]*log(tpred[trialin,i]+eps)
         + (1-target[i])*log(1-tpred[trialin ,i]+eps))
e1[i,trialin] = -(target[i]*log(vpred[trialin,i]+eps)
         + (1-target[i])*log(1-vpred[trialin ,i]+eps))
}
# A = verify(target[-samp], vpred,                   # If ROC is required.
#            frcst.type= "prob", obs.type= "binary")
# roc.plot(A, plot.thres=F,show.thres=F)
}                                                    # End of trialin.
err.trn[trialout] = mean(apply(e0,1,mean,na.rm=T),
         na.rm=T)                                    # Mean over trials.
E1 = mean(apply(e1,1,mean,na.rm=T),na.rm=T)          # Mean over cases.

mod = nnet(input, target,                            # Now get the apparent error.
         size = n.hd, rang = range, decay = wtdecay,
         maxit = max.iter,entropy=T,linout=F)
```

```
err.app = mod$value/n.trn
err[trialout] = .368*err.app + 0.632*E1
}
boxplot(err.trn)
boxplot(err)
hist(err,breaks=50)
plot(err.trn,err)
```

Prediction error.
End of trialout
As in Figure 9 (left).
Ibid (right).
As in Figure 10 (top).
Ibid (bottom).

Acknowledgements Werner Stuetzle, Marina Meila, Peter Hoff, and Hilary Lyons are thanked for numerous conversations and invaluable contributions to this chapter.

References

Bishop, C. M. (1996). *Neural networks for pattern recognition.* (pp. 482). Oxford: Clarendon Press.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Brockman, J. (2006). *What we believe but cannot prove: Today's leading thinkers on science in the age of certainty.* New York: HarperCollins.

Cherkassky, V., & Mulier, F. (1998). *Learning from data: Concepts, theory, and methods*. New York: Wiley.

Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning*, *20*, 273–297.

Devore, J., & Farnum, N. (2005). *Applied statistics for engineers and scientists*. Belmont, CA: Thomson Learning.

Draper, N. R., & Smith, H. (1998). *Applied regression analysis.* New York: Wiley.

Efron, B. (1983). Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, *78*, 316–331.

Efron, B., & Tibshirani, R. J. (1997). Improvements on cross-validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, *92*, 548–560.

Efron, B., & Tibshirani, R. J. (1998). *An introduction to the bootstrap.* London: Chapman & Hall.

Frank, E., Hall, M., & Pfahringer, B. (2003). Locally weighted naive bayes. *Proceedings of the conference on uncertainty in artificial intelligence.* (pp. 249–256). Acapulco: Morgan Kaufmann.

Fu, W. J., Carroll, R. J., & Wang, S. (2005). Estimating misclassification error with small samples via bootstrap cross-validation. *Bioinformatics*, 21, 1979–1986.

Goutte, C. (1997). Note on free lunches and cross-validation. *Neural Computation*, *9*, 1211–1215.

Hall, P., & Maiti, T. (2006). Nonparametric estimation of mean-squared prediction error in nested-error regression models. *Annals of Statistics*, *34*, 1733–1750.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning*. Canada. Springer Series in Statistics.

Haupt, R., & Haupt, S. E. (1998). *Practical genetic algorithms*. New York: Wiley.

Hjorth, J. S. (1999). *Computer intensive statistical methods: Validation, model selection, and bootstrap*. Boca Raton, FL: CRC Press.

Jiang, L., Zhang, H., & Su, J. (2005). Learning K-nearest neighbor naive bayes for ranking. *Proceedings of First International Conference on Advanced Data Mining and Applications (ADMA2005)*. Springer.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, *2*(12), 1137–1143. (Morgan Kaufmann, San Mateo)

Kuk, A. (1989). Double bootstrap estimation of variance under systematic sampling with probability proportional to size. *Journal of Statistical Computing and Simulation*, *31*, 73–82.

MacKay, D. J. C. (1996). Bayesian methods for back-propagation networks. In E. Domany, J. L. van Hemmen, & K. Schulten (Eds.), *Models of Neural Networks III* (pp. 309). New York: Springer. Physics of neural network series.

Marzban, C. (1997). *Local minima and bootstrapping.* Available at http://faculty.washington.edu/marzban/local.pdf.

Marzban, C. (2000). A neural network for tornado diagnosis. *Neural Computing and Applications*, 9(2), 133–141.

Marzban, C. (2004). *Neural Network short course.* Annual meeting of the American Meteorological Society, Seattle, WA. Available at http://faculty.washington.edu/marzban/short_course.html.

Marzban, C., & Haupt, S. E. (2005). *On genetic algorithms and discrete performance measures.* Fourth Conference on Artificial Intelligence Applications to Environmental Science. New York: American Meteorological Society.

Masters, T. (1993). *Practical neural network recipes in C++* (493 pp.). San Diego, CA: Academic.

Masters, T. (1995). *Advanced algorithms for neural networks: A C++ sourcebook* (431 pp.). New York: Wiley.

Rao, J. S., & Tibshirani, R. (1997). *The out-of-bootstrap method for model averaging and selection.* Technical report, May, Statistics Department, Stanford University. Available at http://www-stat.stanford.edu/~tibs/ftp/outofbootstrap.ps

Richard, M. D., & Lippmann, R. P. (1991). Neural network classifiers estimate Bayesian a-posteriori probabilities. *Neural Computation*, *3*, 461–483.

Shao, J. (1993). Linear model selection via crossvalidation. *Journal of the American Statistical Association*, *88*(422), 486–494.

Shao, J. (1996). Bootstrap Model Selection. *Journal of the American Statistical Association*, *91*(434), 655–665.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society* (Series B), *36*, 111–147.

Tian, B. L., Cai, T., Goetghebeur, E., & Wei, L. J. (2007). Model evaluationbased on the sampling distribution of estimated absolute prediction error. *Biometrika*, *94*(2), 297–311, Doi: 10.1093/biomet/asm036.

Zhang, P. (1992). On the distributional properties of model selection criteria. *Journal of the American Statistical Association*, *87*(419), 732–737.

Zucchini, W. (2000). An Introduction to model selection. *Journal of Mathematical Psychology, 44*, 41–61.

Performance Measures and Uncertainty

3

Caren Marzban

3.1 Introduction

Many artificial intelligence algorithms or models are ultimately designed for prediction. A prediction algorithm, wherever it may reside – in a computer, or in a forecaster's head – is subject to a set of tests aimed at assessing its goodness. The specific choice of the tests is contingent on many factors, including the nature of the problem, and the specific facet of goodness. This chapter will discuss some of these tests. For a more in-depth exposure, the reader is directed to the references, and two books: Wilks (1995) and Jolliffe and Stephenson (2003). The body of knowledge aimed at assessing the goodness of predictions is referred to as *performance assessment* in most fields; in atmospheric circles, though, it is generally called *verification*. In this chapter, I consider only a few of the numerous performance measures considered in the literature, but my emphasis is on ways of assessing their uncertainty (i.e., statistical significance).

Here, prediction (or forecast) does not necessarily refer to the prediction of the *future* state of some variable. It refers to the estimation of the state of some variable, from information on another variable. The two variables may be contemporaneous, or not. What is required, however, is that the data on which the performance of the algorithm is being assessed is as independent as possible from the data on which the algorithm is developed or fine-tuned; otherwise, the performance will be optimistically biased – and that is not a good thing; see Section 2.6 in Chapter 2.

Historically, performance has been assessed in terms of scalar measures of performance, i.e., single numbers computed from data, capturing some facet of performance. The mean squared error (Chapter 2), for example, is a measure of accuracy. Other facets are discussed in the next paragraph. More recently, however, the ubiquity of computer graphical techniques has allowed for a more complete and faithful assessment, based on diagrams and other graphic means. It is generally true that a diagram can convey more about performance than a single number can. For this reason, many of the ideas presented in this chapter are based on diagrams.

The most important lesson from this chapter is that performance is a multifaceted concept. For example, consider a dart-board whose center represents the truth. Now, consider the following two players/forecasters: Player A's darts all land in a tightly clustered region to the lower-left of the center. But player B is visually challenged, and so his darts land randomly across the entire dart board. Who is the better player/forecaster? In technical jargon, player A is said to be precise but inaccurate. As unintuitive as it may sound, player B is accurate, though imprecise. He is accurate because accuracy measures the difference between the truth and the *mean* of the forecasts; on the average, player B is right on the bull's eye. So, it is not easy to decide who is better. For a statistician, accuracy and precision are related to bias and variance of the errors, respectively. As we shall see below, these two measures constitute two different components of the mean squared error.

Caren Marzban (✉)
Applied Physics Laboratory and Department of Statistics,
University of Washington,
Seattle, WA 98195-4323, USA
Phone: +(206) 221-4361, fax: +(206) 685-7419;
email: marzban@stat.washington.edu

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

The above example illustrates that it is entirely possible for one forecasting model to outperform another forecasting model in terms of one measure of performance but not in terms of another. That is why it is important to choose the measure of performance thoughtfully. Unfortunately, that is easier said than done. In many situations it is difficult to decide what the "right" measure should be. And that is why most people just pick a measure with some nice and simple properties. Given that approach, the least one should do is to examine *several* measures, and to assure that the measures do not contradict the assumptions of the model or properties of the data.

Finally, since a performance measure is computed from data, it is equally important to assess the statistical variations of its numerical value. After all, it would be a bad thing to fire a forecaster (or to cut funding for the development of some AI algorithm) if the difference between its performance measure and that of a contender is within naturally occurring variations. If the difference is not statistically significant, then one can either keep both, or choose some other criterion for comparing the two, e.g., cost of development. Two articles that address this issue are (Jolliffe 2007) and (Ferro 2007).

3.1.1 The Plan

The goals of this chapter are as follows: (1) Review some basic measures of performance, (2) discuss some of their peculiarities, and (3) point out that a performance value is subject to natural fluctuations, and so, (4) should be accompanied by some measure of uncertainty. But I will also illustrate that (5) uncertainty measures can be misleading and even useless, suggesting that (6) "looking at data", using clever graphs, speaks more to the issue of performance and its uncertainty.

Given the somewhat contradictory nature of these aims, it is appropriate to work somewhere in the twilight zone between uncertainty measures and graphs of data. To that end, most of the "looking" is done by producing graphs of boxplots. This may make it difficult to provide simple answers to questions like "Which forecast/algorithm is better?", but at least the assessments will be less *ad hoc*.

3.2 Details

Suppose we have n pairs of observations and forecasts of hourly temperature, denoted x_i and y_i, respectively. We compute the mean squared error

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2 \tag{3.1}$$

and obtain some number. We have now assessed the performance of the forecasts. What could be simpler? Indeed, the procedure is straightforward, but the problem is that this single number has several defects: (1) It may not mean what we think it does, (2) it does not tell us how to improve the forecasts, and (3) we are uncertain as to how much it would vary for a different sample (but still dealing with the same problem). The following three subsections delve deeper into the meaning of performance measures, whether or not they are diagnostic, and their sampling variations.

3.2.1 Meaning

A performance measure does not always mean what we may think it does. For example, the Pearson correlation coefficient (Devore and Farnum 2005):

$$r = \frac{\overline{xy} - \overline{x}\;\overline{y}}{s_x\; s_y} \tag{3.2}$$

where, in the notation of Chapter 2, the overline refers to sample means, e.g., $\overline{xy} = \frac{1}{n}\sum_i^n x_i y_i$, and s_x refers to the sample standard deviation of x, etc.. It is a measure of the (linear) association between forecasts and observations. If the forecasts are accurate, then r approaches 1; and if the forecasts are inaccurate, then r approaches 0. However, it is tailored to deal with continuous variables, ranging from $-\infty$ to $+\infty$. It does even better if the two variables have relatively symmetric and bell-shaped distributions. Under these conditions, it does indeed tell us how well the forecasts match the observations. If the variables are inherently bound on one side (e.g., the number of tornadoes occurring in a month) or on both sides (e.g., humidity, measured in percent), or with most of the data clustered around one side, then r simply conveys the wrong information about the association between the two variables. It can be near 1 when in fact the accuracy is poor, and it can be near zero when the

accuracy is high in reality. If we insist on using r to measure accuracy, then, it may be better to compute it not for x and y, but for $log(x)$ and $log(y)$, because taking the log of a variable often makes its distribution more symmetric. Of course, this "defect" does not imply that r cannot be computed for any x and y. It can. One can even compute it for binary forecasts and observations, but then it ends up measuring something other than linear association. The point is that we need to be aware of whether the choice of our performance measure is appropriate for the data at hand.

Another example of how a measure may be misleading can be found in the situation where both the forecasts and observations are binary, i.e., a yes/no forecast of a yes/no event. In this situation, one often computes what is called the *fraction correct*, which is the proportion of cases where a "no" observation is correctly forecast as a "no", and a "yes" observation is correctly forecast as a "yes." As intuitive and meaningful as this measure may appear, it has at least one defect: If the events we are trying to forecast are rare, then it approaches 1, quite independently of how accurate or inaccurate the forecasts really are (Doswell et al. 1990; Marzban 1998; Stephenson et al. 2004). Suppose I have an algorithm that predicts whether or not I will win a hundred US dollars on a given day. Even if my algorithm is producing completely random forecasts, the fraction of times it will be correct is still quite high, because of all the times that it will forecast "no win" and will be correct. In fact, it turns out that this rare event situation is one wherein many performance measures become meaningless. What to do? Marzban (1998) and Stephenson et al. (2004) suggest a few measures that are relatively healthy; but caution is always recommended in picking a measure. As the proverb goes "Caution and measure will win you treasure."

3.2.2 Diagnostic

Suppose the previously mentioned problems have been addressed, i.e., we have a performance measure that is appropriate for our problem, and that we are in a non-rare-event situation. To be specific, let us assume that both forecasts and observations have normal distributions. Then, r may be used to assess accuracy. Suppose we get $r = 0.9$. Now what? Well, there is a great deal one can do: assess uncertainty (next subsection), compare with some standard of reference, etc. But there is nothing we can learn from $r = 0.9$ about how we may possibly improve our forecasts. Now, suppose we had instead chosen to assess the performance using MSE (equation (3.1)); and suppose we obtained MSE $= 0.01$. By itself, MSE is again a non-diagnostic measure, but as we shall see in Section 3.3.4, it has a very nice decomposition into two other measures (bias and variance), which in turn suggests a simple transformation of the forecasts that can improve their quality. Diagnostic measures are not too prevalent, and indeed, whether or not they are diagnostic depends on the problem at hand. Two other quantities which are commonly acknowledged as offering diagnostic information are Relative Operating Characteristic (ROC) curves and reliability diagrams, both of which are discussed further in this chapter.

3.2.3 Uncertainty

The following is a fictional account: In the beginning there were 100 pairs of forecasts and observations, and a measure – call it *SS* – which was known to be 0 for random forecasts, and nonzero otherwise, with $SS = \pm\infty$ corresponding to perfect forecasts. The observed value of *SS* for the 100 pairs was 0.13. That is, until someone else computed it on a different set of forecasts and observations for the same problem, and came-up with $SS = -0.05$. After that, every time someone computed *SS* it had a different value, all scattered about zero. Then, someone proposed that the entire data – call it the *population* – is not available to us humans, and that different people are actually observing different *samples* taken from that population. Not only would that explain the variations observed in *SS* values, it would also open the possibility of asking and answering questions like "If the population were made up of a bunch of random forecasts (in which case the true *SS* value would be 0.0) what would be the probability that a single sample would yield an *SS* value of 0.13 (or more extreme)?" In other words, it was known that *SS* values have an inherent spread due to sampling, but the question was if the observed *SS* value was sufficiently far from zero to make it an unlikely outcome. That is an important question if we

are interested in determining whether or not the forecasts have any skill at all (beyond random guessing). The business of answering that question is what statisticians call *Inference* – as in, inferring something about a population parameter (e.g., the true value of *SS*), from a single sample. An alternative, but equivalent question is "What is the typical range of possible *SS* values, if the population really does consist of random forecasts?" Statisticians have a name for that business as well: *interval estimation* – as in, estimating an interval wherein the population parameter may reside, with some confidence. End of story.

The probability of obtaining an *SS* value beyond ± 0.13 of zero, given some assumption about the population (e.g., forecasts are random), is an instance of what is called the p-value. The assumption itself is called the null hypothesis. If the p-value is small, one may reject the null hypothesis in favor of the alternative, $SS \neq 0$. In that case, it would follow that the forecasts have skill, beyond random chance. But if the p-value is large, then the data simply do not provide sufficient evidence to contradict the assumption. In other words, one cannot reject the null hypothesis. In that case, the claim that the forecasts are as good as chance cannot be ruled out. In short, the p-value is what decides whether or not the observed value of a performance measure is sufficiently extreme (compared to what one might expect if the forecasts were random) to call it statistically significant.

The interval-estimation approach is basically a different packaging of the same ingredients. In it, one computes a confidence interval for the population value of *SS*, at some confidence level. The confidence interval can still be used to infer something about the true value of *SS*. For example, if the 95% confidence interval for *SS* does not include zero, then we may conclude that it is unlikely that a sample from a population of random forecasts would lead to the observed value of $SS = 0.13$. As such, we would conclude that the forecasts have statistically significant skill.

Note that, even though both approaches involve a single sample, they are based on a fictional account, or a thought experiment, wherein lots of different samples are taken from the population. In fact, a crucial concept in both of these approaches is the histogram of the *SS* values. This histogram, or rather the distribution of all *SS* values is called the *sampling distribution* of *SS*. In both approaches one must make some assumptions about the population from which the sample is drawn, or about the sampling distribution itself.

There is an alternative that avoids those assumptions. It aims to approximate the sampling distribution with the histogram of lots of *SS* values computed from subsamples taken from the single sample. In other words, one treats the observed sample as a population, and literally performs the aforementioned thought experiment. There is a large class of such *resampling techniques* for parameter estimation, and we have already seen one of the more famous ones in Chapter 2, i.e., the bootstrap (Efron and Tibshirani 1993; Hastie et al. 2001).

In Chapter 2, we employed bootstrap ideas for optimizing the complexity of a statistical model. The procedure called for taking a subsample (called a bootstrap sample) from the original sample, developing a model on the bootstrap sample, and then repeating the procedure for different bootstrap samples drawn from the original sample. The final product of the procedure was a point estimate of the prediction error. Another "outer" bootstrap was then done to approximate the sampling distribution of the prediction error. The sampling distribution can then be used for computing an interval estimate of the prediction error.

But, what if we do not have the luxury of retraining a model on different bootstrap samples? For example, what if the forecasts are generated by an algorithm to which we have no access? An example of this situation is so-called Model Output Statistics (MOS), which the National Weather Service produces (Glahn and Lowry 1972). Or what if it is simply not feasible to run the algorithm producing the forecasts numerous times, e.g., numerical weather prediction models, which are simply too computationally intensive to run 1,000 times? Finally, what if the forecasts are generated by a human? In all of these situations, it is difficult to apply the bootstrap idea to estimate performance. However, one can still apply it to a *given* set of forecasts and observations. The procedure is similar: draw a bootstrap sample from the sample, compute a performance measure (e.g., *SS*), and repeat. The histogram of the performance values approximates the sampling distribution of the performance measure. As such, it tells us something about the possible range of values for the performance measure. The width – or more accurately, the standard deviation – of this histogram

plays an important role in assessing uncertainty. It is called the *standard error* of the performance measure, and shows up in the confidence interval for the measure.

3.2.4 Inference

From the previous section, it should be evident that when I use the term uncertainty I am referring to how much a quantity computed from a single sample may change if it is computed for multiple samples taken from a population. In statistics, one way to formulate that uncertainty is to assume that there is a unique value of that quantity, called the population parameter. Then, the data at hand is treated as a single sample taken from the population; the value of the quantity when it is evaluated on the sample is called the observed sample statistic. There are two standard approaches to quantifying how much the sample statistic varies from sample to sample. One involves building a confidence interval for the population parameter, and the other relies on the probability of observing a statistic more extreme than the observed sample statistic; the latter probability is called the p-value.

Confidence intervals and p-values can be employed for many purposes, but in the context of performance assessment, they are used to decide questions of the type "Is the skill of algorithm X statistically significant?", which simply asks if the forecasts of the algorithm are better than chance. Another typical question is "Is algorithm X superior to algorithm Y at a statistically significant level?" This last question asks if the difference between two algorithms can be attributed to natural/random variations. If so, then one cannot conclude that one algorithm is truly better than the other.

Formulas for confidence intervals and p-values can be found in many statistics text books, including Devore and Farnum (2005). Seaman et al. (1996), and Jolliffe (2007) address many of these same issues. All I will point out here is the connection to typical performance measures. Text books invariably discuss two quantities – the mean of a continuous quantity, and the proportion of a discrete quantity. Classic examples are the mean lifetime of batteries, and the proportion of defective light bulbs, respectively. It turns out that many performance measures fall into one of these types. For instance, a measure called Bias is nothing but the mean of errors,

$$\text{Bias} = \frac{1}{n}\sum_{i}^{n}(y_i - x_i), \tag{3.3}$$

and another measure called probability of detection (Section 3.3.1) is the proportion of cases (out of the cases where an event did occur) where an event is correctly forecast as an event. The central limit theorem tells us that means and proportions generally have normal sampling distributions, and so, confidence intervals for these performance measures are readily available. The general formulas for the 95% confidence interval of the population mean and proportion are

$$\overline{x} \pm 1.96\frac{s}{\sqrt{n}}, \quad p \pm 1.96\sqrt{\frac{p(1-p)}{n}}, \tag{3.4}$$

where $\overline{x}$ and p are the sample values for the mean and the proportion, s is the sample standard deviation, and n is the sample size.[1]

The correlation coefficient (equation (3.2)) is one measure that is a bit difficult to handle, but approximate formulae do exist for its confidence interval. One of the simpler ones is the following 95% confidence interval (Jolliffe and Stephenson 2003):

$$r \pm 1.96\frac{1-r^2}{\sqrt{n}}. \tag{3.5}$$

Standard formulas also exist for the *difference* between means, and the *difference* (or ratio) between proportions. These formulae are important for model comparison, i.e., for deciding if algorithm 1 produces more accurate forecasts than algorithm 2. Again, general 95% confidence intervals are given by

$$(\overline{x_1} - \overline{x_2}) \pm 1.96\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}},$$
$$(p_1 - p_2) \pm 1.96\sqrt{\frac{p_1(1-p_1)}{n_1} + \frac{p_2(1-p_2)}{n_2}}, \tag{3.6}$$

where the labels 1 and 2 refer to the two algorithms. These formulas assume that the two samples are independent; if they are not, then the data are called

[1] These formulas are based on some assumptions and approximations. So, the pedantic reader/researcher is encouraged to check the validity of those conditions (Devore and Farnum 2005).

"paired" and a different set of formulas should be used. They look similar to those in equations (3.4) and (3.6), but with some important differences (Devore and Farnum 2005).

Again, even though these formulas refer to means and proportions (and correlations), in general, they apply readily to verification measures: e.g., Bias qualifies as a mean, and probability of detection qualifies as a proportion. One may wonder why mean squared error does not qualify as a mean? It is certainly the mean of a bunch of things. The problem is that the bunch of things are squared errors and so may not have a normal distribution. The above formulas for means rely on the normality of something – either of the population and/or of the sampling distribution of the mean. That is where ± 1.96 comes from; 95% of the area under the normal distribution is contained within 1.96 standard deviations of its mean. The quantity multiplying the 1.96 in these equations is called the standard error of the respective quantity. Even if the assumptions are violated these formulas work reasonably well as long as the distributions are bell-shaped. For non-bell-shaped distributions (e.g., distribution of counts or squared errors), they do break down. Still, if your choice is between producing interval estimates using wrong formulas, and not producing interval estimates at all, I suggest you go the first way; this way you will have acknowledged that your performance values are subject to variation.

Now, suppose we want to compare the forecasts from some algorithm with random forecasts. This is a rather lenient test, but there are times when it does arise. Although the previous methods can be adapted to answer that question, there is an alternative which is very powerful and smart. It belongs to a class of techniques called permutation and randomization tests (Good 2005a), which are a subset of resampling techniques (Good 2005b). The idea is quite simple: Think of the forecasts and observations as two columns of numbers (categorical or continuous). We can then compute a performance measure, say MSE. Call that value the observed MSE. If the forecasts are random, then we should be able to permute the numbers in the forecast column, and get a similar performance value. By chance alone, some MSE values will be better than the observed one, although most will be worse. The resulting histogram will give a graphic expression of the distribution of all performance values, if the forecasts are random. The area to the left of the observed MSE, is the number of permutations that give better MSE values. Converting that number to a proportion, by dividing the area by the total number of permutations, defines a p-value. It is the probability of obtaining more extreme sample values, under the null hypothesis. If it is below something called an α-level, then we may conclude that the forecasts are not random. Otherwise, they may be. Some typical values of α are 0.05 and 0.01, and *you* (the user) have to choose it. I have discussed this test, because it is a clever test; but I am not delving too deeply into it, because hopefully, our algorithm is doing sufficiently well that we do not need to compare it with the proverbial monkey.

Finally, there is the other resampling technique which we have seen many times – the bootstrap. It can provide an interval estimate of the performance measure. Specifically, take some number of bootstrap samples, and compute the performance measure for each. Then a $1-\alpha$ confidence interval may be computed by taking the $\alpha/2$ and $1-(\alpha/2)$ percentiles of the histogram of performance measures. Again, someone (e.g., the user) has to choose the specific value of α, because it is not determined by any objective criterion.

3.2.5 Why to Not Infer

Having set up all of the above foundation for uncertainty assessment, also called inference, it is time to point out the trouble with it. Consider the confidence intervals given in equation (3.4). Note that all we need to compute these interval is a bunch of sample quantities – we do not need to know much about the population. That is the good news. The bad news is that the intervals shrink as n increases. Why is this bad news? On the one hand, it makes perfect sense for the confidence interval to shrink with increasing sample size – conveying a lower level of uncertainty. On the other hand, and without loss of generality, suppose the value of the performance measure is 0 if the forecasts are random. Then, it is bad news, because no matter how close the observed value is to zero, by simply taking a sufficiently large sample, I can arrange for zero to fall outside of the interval. In other words, with a sufficiently large sample, I can always conclude that my forecasts are not random, even if they really are! Said differently, statistical significance can

be obtained by simply taking a sufficiently large sample. This is a problem with confidence intervals, but p-values have the same problem; it is easy to show that for a sufficiently large sample size, one can arrange for the p-value to be infinitely close to zero. Recall that with an infinitely small p-value, we will always reject the null hypothesis that the forecasts are random.

What do statisticians do? Some go ahead and do all of this anyway, but stay aware of the size of the sample. They keep in mind that if something is found to be statistically significant, it may be simply because the sample size is large, and not because the effect is large. Others do not even do this type of inference; they simply "look" at the data, by selecting appropriate summary measures that can be graphed in useful ways. A compromise is to look at these graphs, but also to display some measure of their uncertainty due to sampling. This is the point of view that I will be taking in the rest of this chapter. Specifically, I will assess uncertainty in performance measures graphically, by "looking" at their sampling distribution. The looking is done through boxplots, and the sampling distribution is obtained empirically, via bootstrapping.

3.3 Special Cases

There is no shortage of performance measures (Jolliffe and Stephenson 2003; Marzban 1998). One reason is that different problems need different measures. That is a valid reason for creating new measures. Another reason is that every measure has some "defect." When people get their hands dirtied with performance assessment, they usually rediscover these defects on their own, and are then inclined to come-up with a measure that is disease free. This process has yielded a plethora of measures, but none that is perfect. The basic reason is that there are just too many criteria that a perfect measure must satisfy; a good measure must be consistent, efficient, sufficient, proper, equitable, insensitive to priors, and more. Here, I will not delve into each of these concepts, but will instead describe some of them in the context of a few currently fashionable measures.

Performance measures come in a variety of styles, depending on the type of forecast and observation. Both forecasts and observations can be categorical or continuous. Whether or not a tornado touches ground is an example of a two-class (binary) categorical observation. Surface temperature is an example of continuous quantity. One might wonder why temperature cannot be treated as a categorical variable, with a large number of categories. One may; but the methodology for handling categorical variables becomes unwieldy for a large number of categories. Also, the various categories in a categorical quantity are assumed to have no specific "order" to them. Variables whose categories have an order are called ordinal, and they need a different methodology than that of categorical data. In AI circles, most common types of observations and forecasts fall into one of the following categories: (1) both forecasts and observations are categorical (here denoted Cat-Cat), (2) both are continuous (denoted Cont-Cont), and (3) observations are binary, but forecasts are probabilistic. These are the only cases I consider in detail in this chapter, although others are discussed in passing.

3.3.1 Cat-Cat

Consider the problem of forecasting hail size (Marzban and Witt 2001). "Size" can be measured in millimeters or it may come in categories. One common set of categories is coin-size, golfball-size, and baseball-size. Those are the observed categories – label them as 0, 1, and 2. If the forecasts fall into the same categories, then the starting point for assessing the performance of the forecasts is the contingency table:

$$\text{C-table} = \begin{pmatrix} n_{00} & n_{01} & n_{02} \\ n_{10} & n_{11} & n_{12} \\ n_{20} & n_{21} & n_{22} \end{pmatrix}, \tag{3.7}$$

where n_{01} is the number of class 0 cases incorrectly forecast as class 1, etc.[2] In my (nonuniversal) convention, the rows correspond to observations, and the columns represent forecasts. This means that the row-marginal $n_{00} + n_{01} + n_{02}$ is the total number of class 0 observations in the data. Similarly, the column-marginal $n_{00} + n_{10} + n_{20}$ is the total number of class 0 forecasts in the sample, etc.

What we call a performance measure is typically a summary measure based on the contingency table. There are many ways we can combine the elements

[2] I use "class" and "category" interchangeably.

in the table to come-up with a summary measure, but most do not represent anything useful. Gerrity (1992) and Murphy (1991) have studied this multi-category problem, coming up with some relatively useful measures. Livezey (2003) has also argued in favor of Gerrity's score, especially if the observations and forecasts are ordinal. The 2×2 case, however, has a longer list of summary measures, and it is better suited for this chapter, as well. For that reason, I will specialize to the 2×2 case only. Note that this does not restrict the forecasts and observations to be binary; it just means that we can treat them as binary. For example, the three-class hail-size problem can be broken down to three two-class problems: class 0 or otherwise, class 1 or otherwise, and class 2 or otherwise. Although this treatment of a three-class problem can lead to some loss of information, the loss is partially compensated by the availability of numerous graphical summary measures that do not have simple three-class (or larger) generalizations (e.g., ROC in Section 3.4.1).

Here is the 2×2 contingency table:

$$\text{C-table} = \begin{pmatrix} n_{00} & n_{01} \\ n_{10} & n_{11} \end{pmatrix}, \tag{3.8}$$

Three common measures derived from this table are the Probability of Detection (POD) (or hit rate), False Alarm Rate (FAR), and False Alarm Ratio (FAO). If the class we are forecasting is the one labeled 1, then

$$\text{POD} = \frac{n_{11}}{n_{10}+n_{11}}, \quad \text{FAR} = \frac{n_{01}}{n_{00}+n_{01}},$$
$$\text{FAO} = \frac{n_{01}}{n_{01}+n_{11}}.$$

Their meaning is evident from these definitions. Each of these three measures does not involve all the elements of the contingency table, and so can be artificially manipulated. For example, by simply forecasting "class 1" for all the cases in the data, one will have POD $= 1$. Of course, in that case one will also have FAR $= 1$, i.e., not good! One relatively healthy measure that involves all the elements is called the Heidke Skill Score:

$$HSS = \frac{2(n_{00}n_{11} - n_{01}n_{10})}{(n_{00}+n_{01})(n_{01}+n_{11}) + (n_{10}+n_{11})(n_{00}+n_{10})}. \tag{3.9}$$

In spite of the unenlightening form of this expression, it is a measure of forecast accuracy relative to random chance (Heidke 1926). In other words, it is zero, if the forecasts are random; positive values imply skill, while negative values also suggest skill but in an anti-correlated sense.

As mentioned above, there is no single good measure for all problems. I have selected these four only for illustration purposes. And what I am about to illustrate is how to obtain the empirical sampling distribution for these measures, i.e., a histogram that conveys the sampling variation of the measures. To that end, I have selected a famous (infamous) data set due to Finley (Wilks 1995).[3]

In the convention of equation (3.7), with "0" labeling "no-tornado", and "1" denoting "tornado", the contingency table for the Finley data is

$$\begin{pmatrix} 2680 & 72 \\ 23 & 28 \end{pmatrix}. \tag{3.10}$$

The values of the performance measures are

$$POD = 0.549, \quad FAR = 0.026,$$
$$FAO = 0.720, \quad HSS = 0.355. \tag{3.11}$$

The numbers are more meaningful if one compares them with what one would expect for them, if the forecasts and observations were not associated at all, i.e., if the forecasts were random. Under the null hypothesis that the forecasts are random, one obtains the following expected contingency table (Devore and Farnum 2005):

$$\begin{pmatrix} 2653.82 & 98.18 \\ 49.18 & 1.82 \end{pmatrix}. \tag{3.12}$$

So, if we were issuing random forecasts, then the performance measures would be:

$$POD = 0.036, \quad FAR = 0.036,$$
$$FAO = 0.981, \quad HSS = 0.0. \tag{3.13}$$

Note that, Finley has drastically better-than-chance POD, and mildly better-than-chance FAR and FAO values. HSS for random forecasts is zero, by the definition of HSS. So, this way, we can compare the forecast performance with those obtained under the null hypothesis.

In passing, I should mention what happens in case the events being forecast are rare. The contingency table in

[3] Note that the sampling distribution of POD, FAR, and FAO are all known, because they are proportions (see Section 3.2.4). But I will pretend that we do not have that information.

(3.10) is typical of what one gets in "rare event" situations. In such a situation most performance measures become pathological (Marzban 1998; Stephenson et al. 2004). For example, consider the measure often called accuracy: it is the proportion of correct forecasts, i.e., $(n_{00} + n_{11})/((n_{00} + n_{01} + n_{10} + n_{11})$. It can be shown that this measure becomes increasingly large the more rare an event is, and this happens quite independently of the true accuracy of the forecasts. For the Finley data, it is $(28 + 2860)/2803 = 96.6\%$. Note that if the forecasts had been a constant forecast of "no tornado" for the entire data set, we would obtain a higher accuracy: 2,752/2,803 = 98.2%! This is an example of how a measure can be misleading when the situation is just wrong.

Returning to uncertainty, so far we have compared the performance measures with those obtained from random forecasts. That is somewhat useful; but now we must also assess the sampling variations. This is where we will turn to bootstrapping. Figure 3.1 shows the empirical sampling distribution of all four measures. I do not aim to summarize these distributions, because the whole distribution carries useful information, but do note that each histogram is centered on the respective values given in equation (3.11), as expected. But, now, we can also get a sense of their sampling variation. In fact, it is the standard deviation of these histograms which is precisely the standard error, appearing in equations (3.4), and discussed in Section 3.2.4.

Let us briefly examine these results before proceeding to the next section. For example, we can see that POD has a wide spread, ranging from 0.3 to 0.8. By contrast, FAR is tightly clustered around 0.036. The

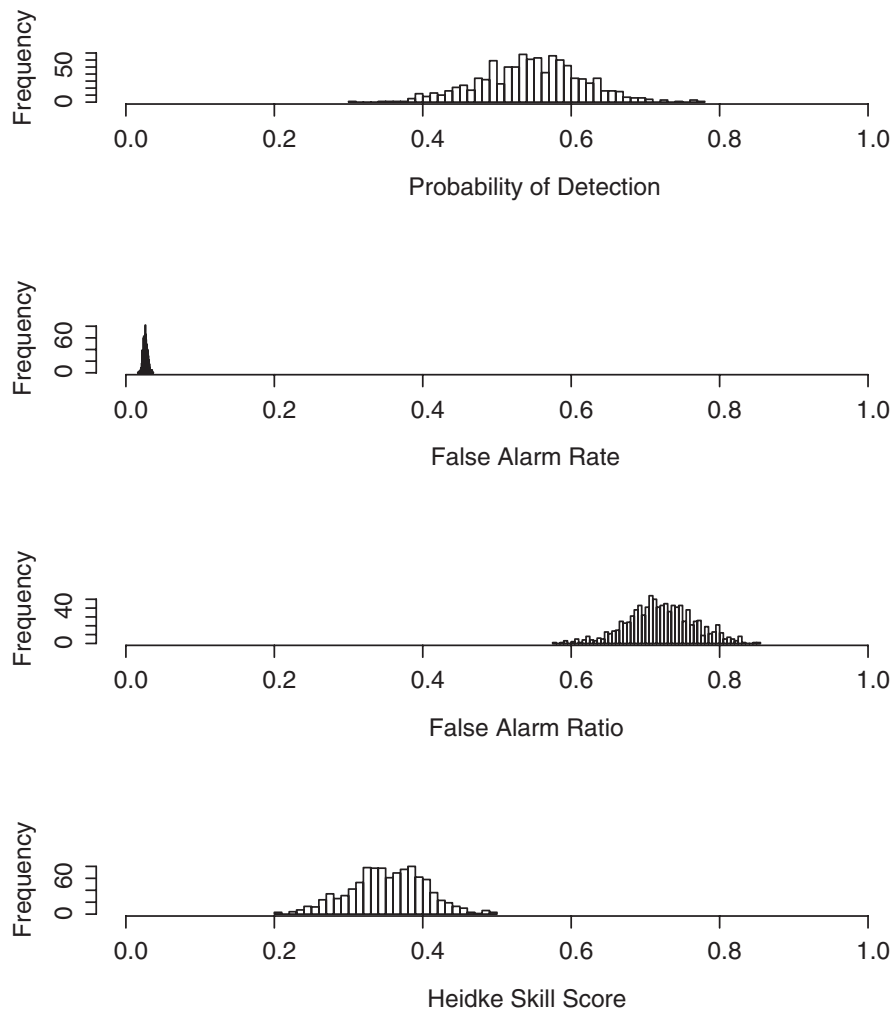


Fig. 3.1 The empirical sampling distribution of four performance measures, displaying their uncertainty, for Finley's tornado data; see text

low value of FAR and its tight spread are due to the large number of nontornadic cases in the data. FAO varies from about 0.6 to 0.8. Finally, HSS can range from 0.2 to 0.5. To reiterate, these numbers are all useful as summary measures, but the histograms speak a great deal more.

The code for producing these histograms is included in Section 3.8. Although the code may have minor bugs in it, the procedure is likely to produce the sampling distribution for any performance measure. It should be relatively simple to revise it to suit your own data, and your own favorite performance measure.

3.3.2 General Framework

Murphy and Winkler (1987, 1992) developed a framework that not only includes the case of categorical forecasts and observations, but is sufficiently general to include any type of forecast and observation. This is a good stage for me to mention it because it suggests how one should handle the non-categorical situations.

First, let us start with the Cat-Cat case, and divide all the elements in the contingency table by n, the total number of forecast/observation pairs in the data. The resulting table, denoted $p(x, y)$, is (an estimate of) what is called the joint probability density of observations and forecasts. In the 2×2 case, for example, x and y are both binary. Given that it is a probability density, it can be decomposed in two famous ways:

$$p(x, y) = p(x|y)\ p(y), \quad p(x, y) = p(y|x)\ p(x), \tag{3.14}$$

where $p(x|y)$ is the conditional probability of observations, given forecasts. The other conditional probability is defined similarly, and the $p(x)$ and $p(y)$ are the unconditional probability of observations and forecasts, respectively.

To make contact with the measures in the previous section, with the same labeling of 0s and 1s, note

$$POD = p(y = 1|x = 1), \quad FAR = p(y = 1|x = 0),$$
$$FAO = p(x = 0|y = 1). \tag{3.15}$$

Evidently, these conditional probabilities are some of the more commonly used measures. $p(x = 1)$ is the *a priori* probability of a "1" (e.g., tornado), period, independently of the forecasts; and $p(y = 1)$ is the probability of forecasting a "1". For the Finley data, they are $(23 + 82)/2{,}803 = 0.037$, and $(72 + 82)/2{,}803 = 0.055$, respectively.

Although, we started this subsection by thinking of the Cat-Cat case, when we write performance measures in terms of the joint probability density of forecasts and observations, $p(x, y)$, then x and y can be anything – categorical or discrete. The next subsection shows how to assess performance of a set of continuous forecasts and observations within the framework of the joint probability density.

3.3.3 Cont-Cont

Now, let us consider a problem in which both forecasts and observations are continuous, e.g., hourly surface temperature (in Farenheit) over Indianapolis between March 2 and June 5, 2002.[4] The best way of assessing the quality of continuous forecasts of a continuous quantity is through a scatterplot (Fig. 3.2, left). If the predictions were perfect, all 5,307 dots would have fallen on the diagonal line. Evidently, the predictions are not perfect. There is some scatter about the diagonal, and there even appears to be a general overprediction of the temperature (this is reflected by the overall right-shift of the dots relative to the diagonal). Both of these issues can be quantified, and the next section shows how.

3.3.4 A Decomposition of MSE

Given n forecasts y_i, and observations x_i, many performance measures take the form of an average of squared errors. The Mean Squared Error (MSE) is one popular example. Such measures have two desirable properties which render them somewhat diagnostic, in the sense that they can convey correcting action on the part of the forecaster or model builder.

[4] The predictions were made by a numerical weather prediction model which at the time was state-of-the-art; it was/is called Advanced Research Prediction System (ARPS).

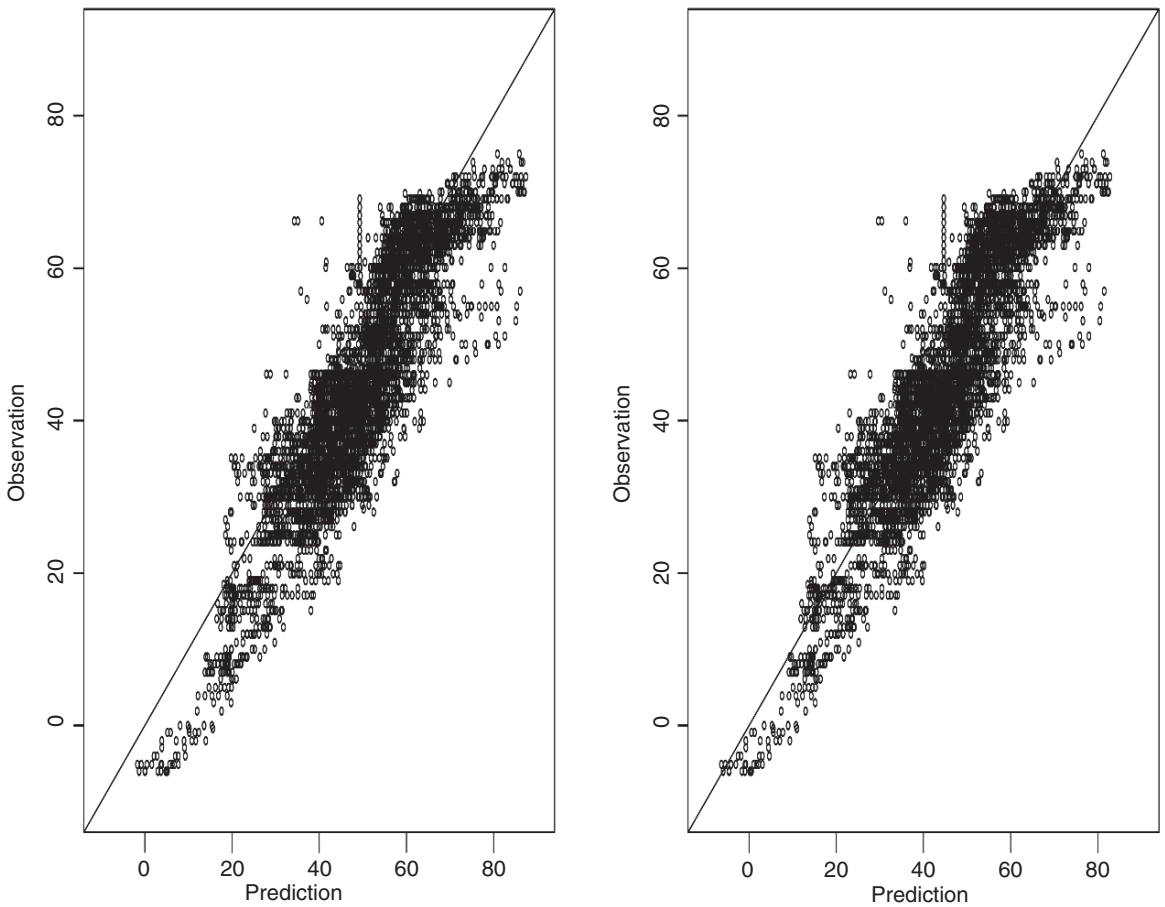


Fig. 3.2 Left: scatterplot of observed and predicted hourly surface temperature (in Farenheit) over Indianapolis between March 2 and June 5, 2002. Right: the same, but for the bias-corrected predictions

The first decomposition is referred to as the bias-variance decomposition:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2 \tag{3.16}$$

$$= \overline{(y_i - x_i)^2} \tag{3.17}$$

$$= \overline{(y-x)^2} - [\overline{(y-x)}]^2 + [\overline{(y-x)}]^2 \tag{3.18}$$

$$= s^2_{y-x} + (\overline{y} - \overline{x})^2 \tag{3.19}$$

$$= \text{Variance} + \text{Bias}^2 \tag{3.20}$$

Again, an overline over a quantity denotes the mean of that quantity. The first term (Variance) is the variance of the errors, and Bias is the difference between the average of the forecasts and that of the observations. This is a useful decomposition because the forecasts can be easily transformed to eliminate the Bias term, without affecting the Variance term. This transformation thereby reduces MSE.

For our temperature example, the value of MSE is 71.45, and it decomposes into a variance of 50.03, and a bias of 4.63. These two numbers quantify the above-mentioned qualitative observations about the scatter and shift of the dots. It is a good practice to report both of them, because they tell two different stories about the quality of the forecasts. Moreover, we can now correct the bias. All we need to do is to shift all of the forecasts by the constant amount 4.63; the resulting MSE will be 50.03 (instead of 71.45). In general, a shift of the forecasts according to $y_i \to (y_i - \text{Bias})$ eliminates the Bias term, while leaving the Variance term unaffected. This reduces MSE down to the variance of the errors; but more importantly, it eliminates bias.

The scatterplot of the bias-corrected forecasts is shown in the left panel of Fig. 3.2. It is clear that the bias correction has worked. However, it is also evident that the bias is worse for higher and lower temperatures, as compared to mid-range temperatures. As such, the bias appears to depend on (i.e., is contingent on) the temperature itself. How are we to assess this particular facet of performance?

The wording of the last paragraph should suggest conditional probabilities. One should consider $p(x|y)$ and $p(y|x)$, because as we have seen they tell different and useful stories. Here, I will examine only the former. Recall that x and y refer to observations and forecasts, respectively (in spite of the fact that the axes appear switched in Fig. 3.2). $p(x|y)$ is the probability of an observed temperature, for a given forecast temperature. However, given that x is now a continuous variable, it is more convenient to compute the conditional mean (or expected value), $E(x|y)$.

How are we to estimate this quantity from data? Recall that probabilities are estimated as fractions; e.g., $p(y = 32)$ is estimated as the fraction of cases in the data that have $y = 32$. In theory, though, with continuous variables, each value of y will appear only one time in the data, in which case every value of y will have the same probability. That is not a terribly good estimate of the probability. So, what one does is to bin the data. In other words, we will partition the predictions into several intervals, and compute the fraction of cases that fall within each interval. In other words, we estimate $p(y)$ with a histogram of y. Note that the size of the bins is important, because it determines the number of cases falling within each interval. With a terribly small bin size, we are back to having a single case in each interval; and with a huge bin size, we will not be able to see how $p(y)$ depends on y. So, pick a reasonable bin size.

Returning to the scatterplot, we now know how to estimate $E(x|y)$ from data: bin the y's (the forecasts), and compute the average of the observed temperatures for each bin. The resulting means can be plotted on the vertical axis, as a function of the forecasts. Such a plot is called a *reliability plot*, and assesses conditional bias.[5]

But, given that we are now talking of computing a (conditional) mean, a natural issue is the sampling variation of that mean. In other words, how much do we expect the conditional means to vary from sample to sample? As the reader may anticipate, the bootstrap approach answers that question. I performed 1,000 bootstrap trials on the conditional mean of observed temperatures. Figure 3.3 shows the "reliability plot", and much more (which is why I use quotes). It shows boxplots of the conditional mean of the observed temperature, given the forecasts.[6] In this way, we can not only visually assess the reliability of the forecasts – by comparing the position of the boxplots relative to the diagonal – we can also say something about uncertainty. For example, relative to a midrange forecast (say 42), a forecast of a high temperature (say 77) is accompanied by a larger variation in the corresponding observed temperatures. In short, the high-temperature forecasts are less reliable, because they are generally lower than the observed temperature; but we should be less certain about that unreliability, because the boxplots are larger.

It is time for a warning. It is tempting to monitor the overlap of the boxplots with the diagonal line. One may be tempted to conclude that if a boxplot extends to the diagonal line, then the corresponding forecasts are reliable. However, this conclusion would be unwarranted. Here is why: By employing boxplots, we are effectively looking at the sampling distribution of the conditional mean. Recall that the width of the sampling distribution decreases with sample size (Section 3.2.4). As such, the same problems that plague the confidence interval also effect these boxplots. Here is a facetious way of making the point: Given the inverse relationship between sample size and the width of the sampling distribution of the mean, we could guarantee reliable forecasts by shrinking the sample size, because then the boxplots are guaranteed to cover the diagonal!

So, then, what is the purpose of the boxplot version of the reliability plot? As we said before, the purpose is to give a visual means of assessing sampling

[5] In the literature, one usually sees reliability plots when dealing with probabilistic forecasts of binary events; Section 3.4.2. Note, however, that we have just shown that the Murphy-Winkler framework allows for a similar plot for continuous observations and forecasts, as well.

[6] The horizontal bar in the box marks the median of the conditional means; the lower and upper side of the box denote the first and third quartiles, and the whiskers show the minimum and maximum of the conditional means.

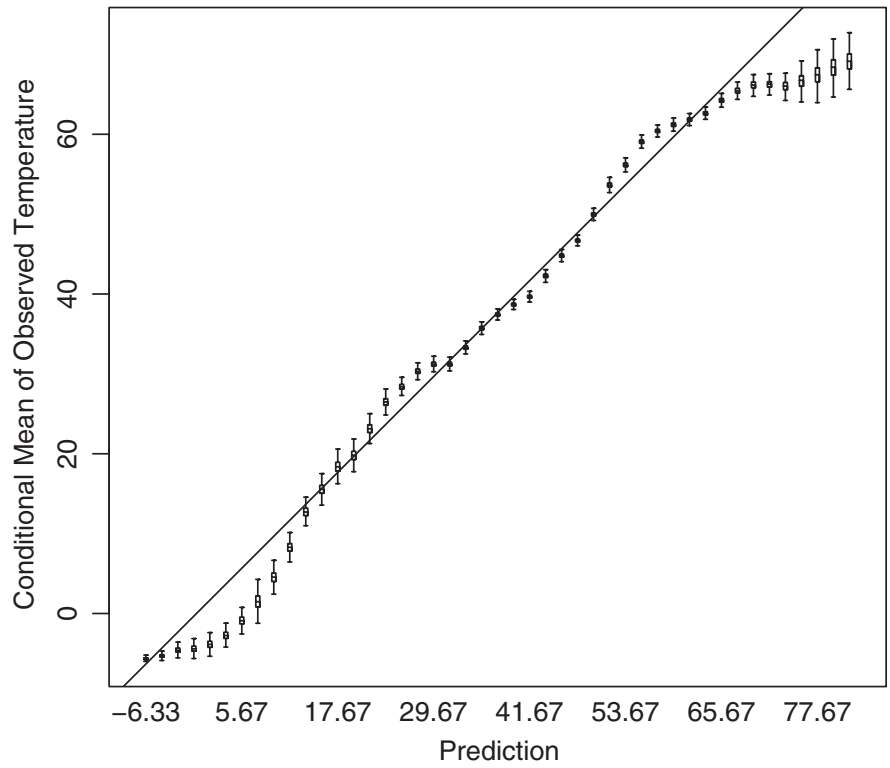


Fig. 3.3 The "reliability plot" for the Indianapolis temperature forecasts. The traditional reliability plot would show only the conditional mean of the observations along the vertical axis, and not the boxplots. Hence the quotes around "reliability plot"

variations. For example, Fig. 3.3 is telling us that for midrange temperatures, the forecasts are reliable and also do not vary much; forecasts in the high temperature range are not reliable and vary more, as well. In summary, where the forecasts are reliable (i.e., close to the diagonal), they are also certain (small boxplots), and vice versa. This is the kind of behavior one should consider "good". The other purpose is that boxplots convey something about the shape of the sampling distribution itself. For example, if the horizontal line is not in the middle of the box, then one can conclude that the distribution is skewed. That kind of information can be useful in correcting the forecasts. And speaking of correcting the forecasts; it is clear that the prediction algorithm in Indianapolis would be better by underforecasting temperatures above 60 F, and in between −5 F and 15 F. By how much? The answer is in Fig. 3.3.

To recapitulate, recall that we decomposed MSE into bias and variance, and then ended-up discussing conditional bias (i.e., reliability) and its sampling variation at great length. All because bias – conditional, or not – is an important, intuitive, and diagnostic measure of performance. But what happened to the variance term? We are not paying too much attention to it, because it is not a diagnostic quantity. Estimating it will give us a measure of performance (and we did that, above), but there is not much we can do to the forecasts to improve their performance in terms of variance. I have included in Section 3.8 a code that does the bias-variance decomposition, and should generate the analog of Fig. 3.3 for any other data set involving continuous forecasts of continuous observations.

3.4 Probabilistic Forecasts of Binary Events

Now consider the case where the observations are binary – $x = 0, 1$ – but the forecasts, y, are probabilistic, say the probability of $x = 1$. The conditional probabilities take the form $p(x = 1|y)$, $p(y|x = 0)$, $p(y|x = 1)$, and the unconditional probabilities are $p(y)$ and $p(x = 1)$. The $x = 0$ elements can be written in terms of these. Note that all of these probabilities involve the forecasts y, and so say something about forecast quality, except the last one – $p(x = 1)$;

it refers only to the prior probability of having an event, independently of the forecasts; in weather circles, it is called the climatological probability of an event.

To avoid confusion, let me emphasize that there are two different sets of probabilities in this section: one is the probability of an event, e.g., probability of tornado. The other is the joint probability density of forecasts and observations. The former is a forecast probability, while the latter provides a framework for assessing the quality of the former. When in the previous paragraph I said y is the probability of $x = 1$, that does not mean $y = p(x = 1)$. More accurately, y is the conditional probability of $x = 1$, given any information that has gone into making the forecasts. For example, $y = p(x = 1|\text{data})$, or $y = p(x|\text{forecaster's knowledge})$, $y = p(x|\text{some predictor})$. The special case of $y = p(x = 1)$ corresponds to when the forecasts are based only on the *a priori* (or climatological) probability of $x = 1$.

3.4.1 Avoiding Probabilities

Before proceeding with the assessment of probabilistic forecasts, let me mention that it is possible to avoid probabilities altogether! This can be done by categorizing the forecast probability; you simply pick a number between 0 and 1 (because probability lives in that range) and then say that any forecast probability less than that number belongs to class 0, and any forecast larger than that threshold belongs to class 1. In this way, one can reduce a continuous quantity like probability into a binary one, and the procedure described in Section 3.3.1 can be employed. In order to not restrict oneself to a fixed threshold, one can compute some performance measure (e.g., HSS), and plot it as a function of the threshold. These curves usually have a single maximum, and so, the probability threshold at that maximum is a useful number to note. Marzban and Stumpf (1998) produce plots like this for numerous performance measures.

A variation on that theme is the idea behind the Relative Operating Characteristic (ROC) curve; see Fawcett (2006), Marzban (2004) and the references therein. At each value of the threshold, one can compute a contingency table. From the contingency table, one can compute not one performance measure (like HSS), but two – POD and FAR. The set of all thresholds manifests itself as a "curve" on a plot of POD vs. FAR. This curve is called the ROC curve. If the ROC curve is diagonal, then the forecasts are as good as random, and the more the curve bows to the upper left, the better are the forecasts. The ROC curve constitutes a decent measure of performance, even for probabilistic forecasts because it expresses performance without a specific reference to a unique threshold. A specific choice of the threshold would call for knowledge of the costs of misclassification which are user dependent. In this way, an ROC diagram offers a user-independent assessment of performance.

Keep in mind though, that an ROC diagram does not present a complete view of all the facets of performance. Given that it involves POD and FAR, a look at equation (3.15) reveals that the ROC diagram captures only the components of the joint distribution that are contingent on observation. The components conditioned on the forecast are left out of the picture. We will return to this, below, when we look at the reliability plot; it is conditioned on forecasts. In general, it is a good idea to assess performance using "complimentary measures" that are conditioned on forecasts and observations, respectively, e.g., ROC plots and reliability plots.

In Chapter 2, we came across ROC plots. There, we were occupied with model development and selection; we used re-sampling to produce a number of different ROC curves for a given model structure, thereby obtaining a visual sense of the spread in the curve. In this chapter, we are dealing with a fixed set of forecasts, but we can still employ the bootstrap to get a sense of the sampling variation of the ROC curve. Although, we can use equation (3.4) for placing errorbars on POD and FAR, it is more instructive to view the entire distribution of POD and FAR. Macskassy and Provost (2004) provide a thorough account of how one should produce confidence bands for ROC curves. Here, though, I will again turn to bootstrap for help.

To illustrate, I chose a problem where a neural network is designed to distinguish between a solid sphere configuration of the Earth and another configuration with a cavity (e.g., oil reservoir) just under the surface, based on the gravitational field around the Earth. In that problem I had 1,000 pairs of forecasts and

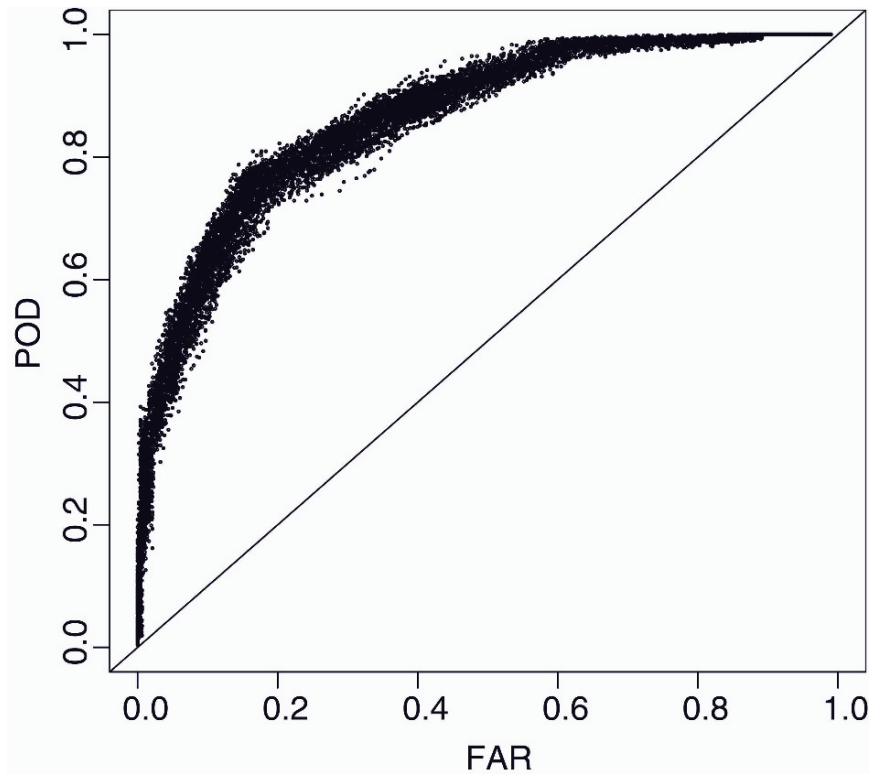


Fig. 3.4 The ROC diagram of 100 bootstrapped ROC "curves", effectively displaying the full sampling distribution of POD and FAR. The traditional ROC diagram would involve one curve, or a set of curves. However in the bootstrap scheme, it makes more sense to plot all the dots, without connecting them with lines

observations. I drew 100 bootstrap samples from the data, and plotted the ROC curves. However, instead of drawing the curves, I plotted only the points corresponding to the aforementioned thresholds. The result is shown in Fig. 3.4. This diagram effectively shows the empirical sampling distribution of the ROC "curve". This type of plot clearly shows the full distribution of possible ROC variations due to sampling. In this particular case, the conclusion is that my classifier was definitely capable of performing the classification, in a manner that can be called highly statistically significant, because the ROC cloud does not cover the diagonal line. The code producing this plot is included in Section 3.8.

Let me end this section by reminding the reader that if one is comparing two ROC clouds from two different models – perhaps, for the purpose of choosing the better model – then one should wonder if the data between the two models are somehow paired. If the data are paired, then one should look at the difference between ROC curves. Given that an ROC curve involves two quantities (POD and FAR), the difference can be taken either in the POD direction, or the FAR direction. The choice is not a priori obvious, and so, it may be worth looking at both.

3.4.2 Avoiding Probabilities, but Less

In the previous section, even though we were dealing with probabilistic forecasts, we converted the probability to a binary quantity, and thus, assessed performance using tools designed for binary forecasts and observations, e.g., 2×2 contingency table, and the ensuing POD and FAR. But note that converting the forecasts to binary is tantamount to binning the forecasts into two intervals – less than threshold, and greater than threshold. Why not bin the forecasts into finer intervals, say $K = 10$? We can, and that will convert the forecasts into a categorical variable with 10 categories. So, may we then use a 2×10 contingency table to assess performance? Again, we could, but then we would be throwing away a great deal of information in the ordered nature of the 10 categories. By binning

probabilities into multiple intervals, one ends-up with an ordinal variable. But that is exactly the situation in which we found ourselves back in Section 3.3.3 when we were dealing with continuous forecasts and observations. There, we binned both the forecasts and observations into intervals. So, why can we not do the same thing? We can, and we will. In other words, we will bin the forecast probabilities into some number of intervals (conventionally 10).

In fact, we can proceed with using the same machinery we developed for continuous forecasts. For example, we can compute the reliability of the forecasts, $E[x|y]$. Again, recall from Section 3.3.4, that we like this performance measure because it is diagnostic, i.e., it tells us how to fix the forecasts. Also, recall that it is estimated by taking the average of observations (i.e., x values), for a given range of forecasts (i.e., y values). But, since we are considering binary observations (i.e., $x = 0, 1$), what does it mean to take the average of the x values? It is easy to show that the average of a bunch of 0s and 1s is nothing but the fraction of 1's. In other words, $E[x|y] = p(x = 1|y)$. So, a reliability plot becomes nothing more than a plot of an estimate of $p(x = 1|y)$ (i.e., observed fraction) versus a set of binned y values. Needless to say, reliable forecasts are those for whom $p(x = 1|y)$ is equal (or at least close) to y itself. In other words, the reliability "curve" should fall on, or around, the diagonal line on the plot of $p(x = 1|y)$ vs. y; for a deeper discussion of reliability diagrams, and a more sophisticated version called the attributes diagram, see Wilks (1995). Hamill (1997) discusses the multicategory generalization of reliability diagrams for probabilistic forecasts.

But then the question of sampling variation arises again. How much does the reliability curve fluctuate due to sampling? Again, given that we are dealing with proportions, we could develop a formula, but let us use the bootstrap procedure again. Figure 3.5 shows the result for the forecasts I mentioned in the context of the ROC plot, in the previous Section. The code for generating these results is included in Section 3.8. The set of boxplots present a visual summary of the conditional (on forecast) sampling distribution of the fraction observed. In this particular example, the good news is that the boxplots generally do fall along the diagonal. The bad news is that some forecasts are generally below the diagonal; e.g., at 0.1 and 0.5. Also, mid-range probabilities (say, 0.3–0.7) are generally associated with longer boxes and whiskers, and so, are generally associated with more uncertainty in the corresponding observed fraction.

What are we to do with all this information? Given that we are dealing with conditional bias, we already know how to improve the forecasts in this regard; see, Section 3.3.4. When the boxplot falls below the diagonal line, one says that there is overforecasting. For example, when the forecast probability is 0.1, the fraction observed is less than that number. To fix this problem, one should issue less 0.1 forecasts. Underforecasting is remedied in a similar way. What do we learn from the boxplots? As advocated all along, they give us a visual display of uncertainty. So, for example, we cannot be too certain about whether or not the forecasts at 0.45 are reliable, at least relative to the forecasts at 0.95. As such, we should be less inclined to correct our 0.45 forecasts.

3.4.3 Another Decomposition of MSE

We have seen that MSE decomposes into variance and bias. We have also seen that conditional bias is related to reliability. Is there a decomposition of MSE that involves reliability? And if so, what is in the rest of the decomposition?

For probabilistic forecasts of binary events, MSE has another decomposition motivated by the factorization of the joint probability density, equation (3.14). Recall from the previous section that the probabilistic forecasts y_i can be binned into K non-overlapping intervals. For instance, for K = 10 the intervals could be 0–0.1, 0.1–0.2, . . . , 0.9–1.0. Let these intervals be denoted by $f_k, k = 1, 2, \ldots, K$. Then the sum appearing in MSE can be re-grouped as follows (Murphy and Winkler 1987, 1992; Wilks 1995):

$$\begin{aligned} MSE &= \frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2 \\ &= \frac{1}{n}\sum_{k=1}^{K} n_k(f_k - \overline{x_k})^2 - \frac{1}{n}\sum_{k=1}^{K} n_k(\overline{x_k} - \overline{x})^2 \\ &\quad + \overline{x}(1 - \overline{x}) \\ &= \text{Reliability} - \text{Resolution} + \text{Uncertainty}, \end{aligned} \tag{3.21}$$

where $\overline{x_k}$ is the mean of the observations (i.e., the observed fraction) in the kth interval.

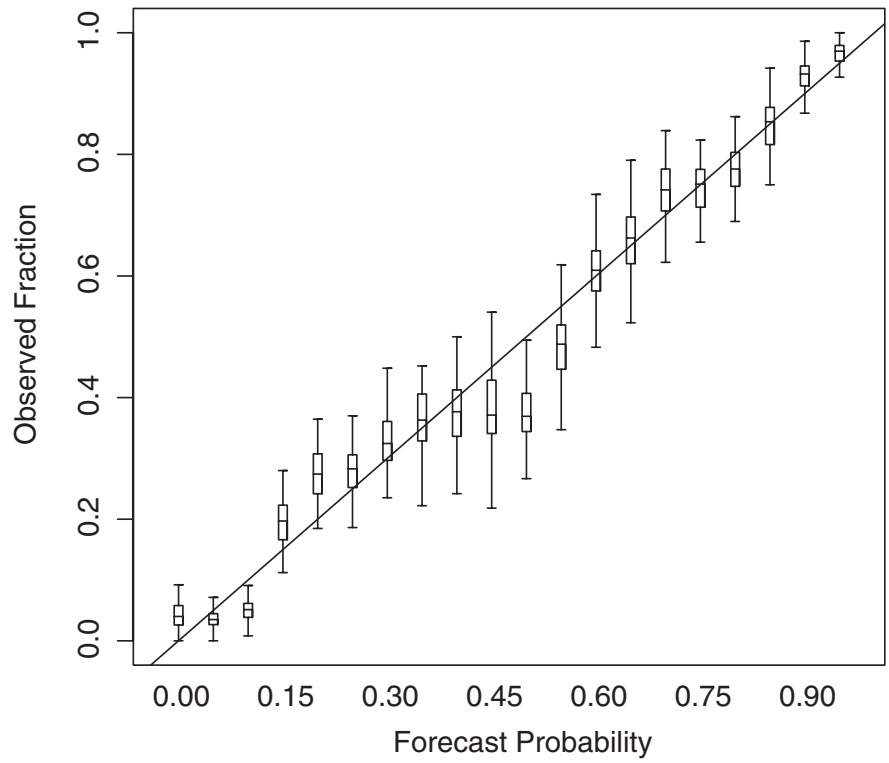


Fig. 3.5 The reliability diagram for the probabilistic forecasts from the gravity neural network

So, the answer to the first question asked at the beginning of this section is in the affirmative. It is easy to see why the first term is called reliability: it is a weighted average of the difference between the reliability curve and the diagonal line in the reliability diagram. This term can be thought of as a scalar summary measure of the reliability diagram. Do note, however, that in looking at a reliability diagram one talks about "*high* reliability" as a good thing; but given equation (3.21), good forecasts correspond to a *small* reliability term in the decomposition.

The second term gauges the amount of spread of the observed fractions in the kth interval ($\overline{x_k}$) about $\overline{x}$, which for $x = 0, 1$ is nothing but the overall fraction of 1's, i.e., the prior probability of an event. According to the decomposition, we would want this term to be as large as possible, because of its negative sign.

The last term refers only to the observations, and so cannot be affected by the forecasts. So, it does not enter the picture of performance assessment. It is called uncertainty, but technically, it is the variance of a Bernoulli variable, i.e., a random variable taking only 0, 1 values.

I will not include in this chapter the results of this decomposition on our gravity example. But suffice it to say that we can use the bootstrap approach to produce histograms that represent the sampling distribution of the first two terms appearing in equation (3.21).

3.4.4 Beyond Reliability

As we mentioned, $p(x, y)$ has two useful factorizations. For binary events, the quantities of interest are $p(x = 1|y)$, $p(y|x = 0)$, $p(y|x = 1)$, and $p(y)$. The first one we have already addressed – it measures reliability. The last one, is called *refinement*, and its estimate would be the histogram of all the forecast probabilities. So, it tells us something about the range of forecasts probabilities, and their frequency. If the forecast probabilities are all clustered around 0 and 1, then, one speaks of the probabilities as not being refined. By contrast, if the probabilities vary over the full range between 0 and 1, then we may consider the probabilities as being refined. I will skip refinement, here, and move on to the more interesting factors.

The other two factors, $p(y|x = 0)$ and $p(y|x = 1)$ can be visualized with histograms as well – the conditional histogram of the probabilities for only

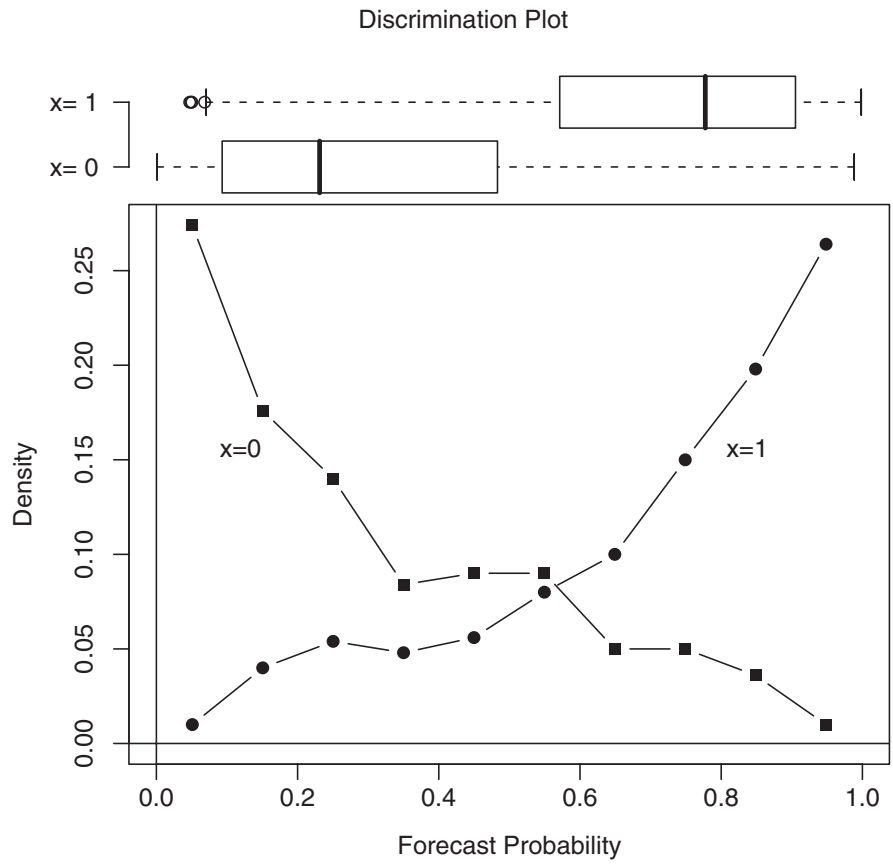


Fig. 3.6 The discrimination plot for the probabilistic forecasts from the gravity neural network

non-events ($x = 0$), and for events ($x = 1$), separately. When these two histograms are placed on a single plot, the resulting diagram is called a discrimination plot. It exhibits the extent to which the forecasts discriminate between the classes. Ideally, there should be many low-probability forecasts for $x = 0$, and many high-probability forecasts for $x = 1$. Figure 3.6 shows the discrimination plot for our gravity example. Recall that y is the forecast probability that the neural network produces, and x denotes the true class (0/1). Given the separation of the two histograms, we can conclude that the forecasts are quite discriminatory. The code for generating this figure (and Fig. 3.7) is included in Section 3.8. The boxplots on the top of the figure summarize the distribution of the two histograms; again, the lack of an overlap between the boxes shows that the neural net can properly discriminate between $x = 0$ and $x = 1$ classes.

As you may suspect, the question then is how to quantify the uncertainty in the two histograms? To that end, we return to our bootstrap approach. I took 500 bootstrap trials, and computed $p(y|x = 0)$ and $p(y|x = 1)$ for each trial. Again, the business of "paired data" surfaces itself; because we are trying to compare $p(y|x = 0)$ with $p(y|x = 1)$, but they are both estimated from the same bootstrap sample. As such, they are not independent, and so, we should be looking at their difference, and asking if the boxplot of the difference includes the number zero. Alternatively, we can examine their ratio, and ask if the boxplot of the ratio includes the number 1. Here, I will look at the difference – the absolute value of the difference, to be specific.

An ideal discrimination plot consists of two well-separated histograms. Consequently, the absolute value of the difference between the two histograms would display a U-shaped pattern (not necessarily symmetric), with the bottom portion of the U touching the x-axis. Of course, due to sampling variations, in practice things do not look that simple, and that is why we look at the boxplot of the difference. If the boxplots in the bottom portion of the U include zero, then that is a good thing. Meanwhile, we would want to see the boxplots for the extreme forecast probabilities

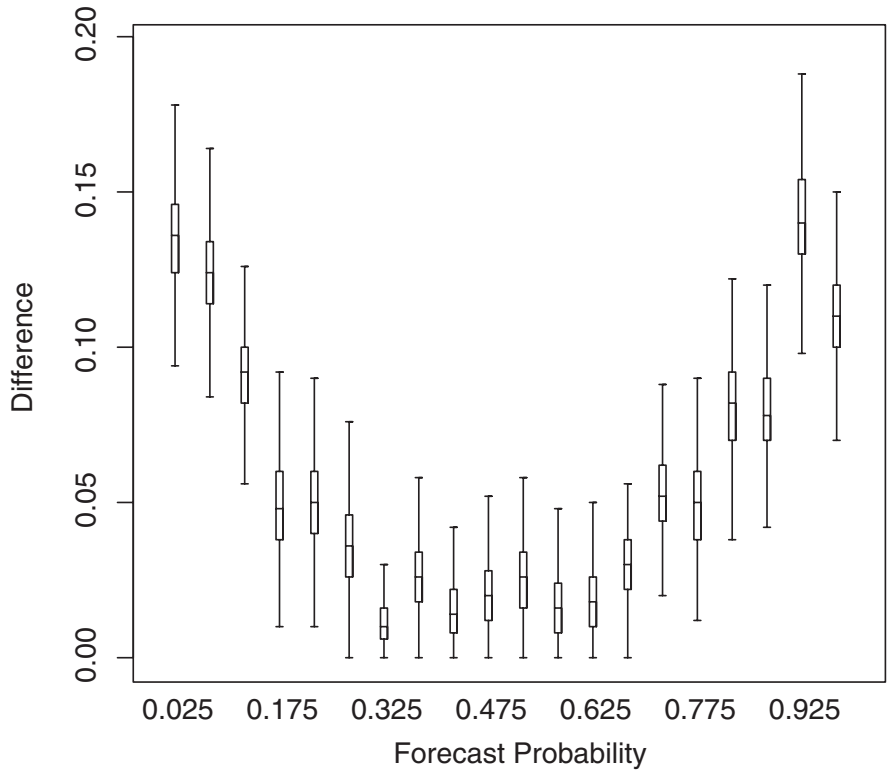


Fig. 3.7 Boxplots of the sampling distribution of the difference $|p(y|x = 0) - p(y|x = 1)|$, where y denotes the forecast probability (plotted along the x-axis). What we look for, here, is whether or not the boxplots extend down to the horizontal axis at 0

far away from zero. Figure 3.7 shows the result of performing 500 bootstrap trials, and boxplotting the difference $|p(y|x = 0) - p(y|x = 1)|$ for the gravity example. The number of intervals along the horizontal axis is 20. Evidently, within the variations reflected in the boxplots, the forecasts from the neural net are discriminatory – almost ideally discriminatory. As such, there is no reason to try to improve the forecasts as far as discrimination is concerned. On the other hand, there would be good reason to examine how the neural net was trained, if the boxplots in the bottom portion of the U did not extend to the x-axis, or if the boxplots for the extreme forecast probabilities did extend to the x-axis.

In summary, the quality of probabilistic forecasts of binary events can be assessed by the joint distribution of forecasts and observations. This joint distribution can be factored into components which can be readily presented as three diagrams – reliability (or calibration, or conditional bias), refinement, and discrimination diagrams. In interpreting these diagrams, it is important to take into account sampling variations. Resampling techniques (e.g., bootstrapping) can be employed to literally view the effect of the sampling variation on the various diagrams.

3.5 Final Comments

3.5.1 Beyond Quality

Murphy argued that the goodness of a model has three different components: quality, consistency, and value (Murphy 1993). Everything we have discussed above has been in regards to quality.

The issue of consistency arises because certain situations allow for, and even encourage, a forecaster to change his/her forecasts. The term "situation" refers to a combination of the choice of the performance measure, and the way in which the measure is allowed to feed back on the forecasts. Some related concepts are equitability (Gandin and Murphy 1992; Gerrity 1992; Marzban and Lakshmanan 1999) and hedging (Murphy and Epstein 1967). An example will serve

to demonstrate the concept. Suppose a forecaster is in possession of a box that is guaranteed to produce high-quality, high-reliability, and high-value forecasts. But suppose this forecaster's boss evaluates him on the basis of his MSE. It is easy to show that the forecaster can actually improve his evaluations by not issuing the probabilities that the box produces, but issuing probabilities that are closer to the *a priori* (climatological) probability. In the long run, his MSE will be lower than if he had followed the advice of the box. This forecasting system is referred to as *improper*, in that it encourages the forecaster to issue forecasts that are not necessarily best in any useful sense.

The above paragraph refers to a "forecaster", suggesting a human element present in the "situation". However, even without a human element present, e.g., an AI algorithm producing forecasts autonomously, it is still possible for the developer of the algorithm to have produced an algorithm that is implicitly inconsistent. After all, it does not take much to insert a line or two at the end of an AI algorithm that simply hedge the outputs in one way or another. In most cases this type of postprocessing is justified, but it is good to be aware and concerned over the possibility of developing an inconsistent forecasting system – AI-based, or not.

As for the value of forecasts, the term "value" refers to economic value. It is important to realize that assessing the quality of forecasts is usually independent from assessing their economic value. Roebber and Bosart (1996) present a real-life illustration of how quality and value are entirely different facets of performance. This point becomes obvious as soon as one notes that the construction of performance measures does not involve any information regarding the cost of taking action (e.g., moving the car into the garage), and the loss associated with not taking any action, if an event (e.g., hail) does occur. Whether or not one should take action is a matter of decision making, and not of the quality of the forecasts. In order to assess the value of the forecasts, it is possible to set-up economic models that quantify the aforementioned costs and losses. One of the simpler models leads to a single scalar measure of forecast quality that ends-up depending only on the cost/loss ratio. In that simple case, one can actually plot the performance measure as a function of cost/loss ratio. Although these plots are still measures of forecast quality, they go beyond forecast quality because they do convey information that is more useful for decision making. Richardson (2000) and Wilks (2001) report on several such value curves. It would be interesting to see how these curves respond to sampling variation.

3.5.2 Probabilistic Forecasts of Continuous Events

The observed event considered in the previous section was binary ($x = 0, 1$), and the probability was for $x=1$ to occur. What if the observed quantity is continuous, but we still want to produce probabilistic predictions? We can do what we did in Section 3.3.3 when handling continuous observations – categorize them into multiple intervals; then, we can assess the probabilities in each category. A full discussion of that analysis will take us beyond the scope of this chapter. I refer the reader to the following references. I have already mentioned Hamill (1997) who considers the multicategory generalization of the reliability diagram. Roulston and Smith (2002) approach the problem from an information theoretic point of view, but their emphasis is on finding a good scalar measure of performance. Gneiting and Raftery (2005) also discuss that problem, and put forth the Continuous Ranked Probability Score (CRPS) as a measure that satisfies many of the requirements of a good performance measure. A more general and diagnostic framework is developed in Gneiting et al. (2007).

3.5.3 Ensembles

We have learned how to assess the quality of probabilistic forecasts. But that presumes that the forecasts are probabilistic. Many algorithms naturally produce probabilities as their output (e.g., neural networks), while other algorithms can be coaxed into producing probabilities (e.g., k-nearest-neighbor); see Chapter 2. However, there are situations wherein a handful of forecasts are available for a given observations. This is the case in what is commonly known as ensemble forecasting systems. An ensemble is composed of some number of algorithms, each typically producing a single deterministic forecast. Wilson et al. (1999)

devise a framework for assessing the quality of such forecasts. Gneiting et al. (2005) introduce a scheme for calibrating the forecasts (i.e., rendering them reliable) by minimizing the CRPS mentioned above. Raftery et al. (2005) reach the same goal but by "combining" the deterministic forecasts from each ensemble member to produce a single, reliable probabilistic forecast.

3.5.4 Spatial Forecasts

Another situation that I have not addressed here is the evaluation of predictions that have a spatial structure. The best example is an algorithm that produces forecasts of precipitation (i.e., rain) across some region. We all know from weather maps shown on television that weather has a spatial structure. A typical weather map may have a dominant cluster of rain in one region, scattered showers in another region, mixed with anything else one can imagine. An algorithm may produce forecasts that resemble the actual weather map, but perhaps place the clusters in the wrong place. Or the clusters may be in the right place, but with the wrong amount of rain. The clusters may be of the wrong size, or they may not have the correct cluster structure at all. You can see that there are many more facets to the assessment of spatial forecasts. And when we try to make the forecasts probabilistic, the problem is exacerbated. Some recent attempts to solve these problems have been proposed by Baldwin et al. (2002), Brown et al. (2004), Casati et al. (2004), Du and Mullen (2000), Ebert and McBride (2000), Nachamkin (2004), Marzban and Sandgathe (2006, 2007), Marzban et al. (2008), and Venugopal et al. (2005).

3.6 In Closing

At the start of this chapter, I may have given the reader the impression that rigor and unambiguity will play an important role in what is to be discussed. Now, at the end of the chapter, it may all seem very convoluted and confusing. Part of that may be attributed to my own confusions; but a hopefully larger part can be attributed to the fact that the topic of performance assessment is highly complicated in and of itself. And, then, assessing the uncertainty of performance measures makes matters even worse. The fact is that in dealing with complex issues, there are no simple answers. That is why everything I have presented here ends-up being somewhat qualitative, in spite of the equations and the graphs. However, what I hope the reader will take away from this chapter is an appreciation of the complexity of the issues, and a set of tools designed to shed some some light on the matter.

3.7 Computer Code

```
#####################################################################
# This code produces the sampling distribution of several performance
# measures computed on Finley's tornado data.
#####################################################################

rm(list=ls(all=TRUE))                                    # Clears things up.
#####################################################################
# Function for computing the Heidke skill statistics for a contingency table.
        heidke function(dim, ct)
        {           ct = ct/sum(ct)
        margin.table(ct,1)
        margin.table(ct,2)
        a = b = 0
        for(i in 1:dim){
        a = a + ct[i,i]
```

```
        b = b + margin.table(ct,1)[i]*margin.table(ct,2)[i]
        }
        if(b!=1)
        (a-b)/(1-b)
        else
        0
        }
################################################################
x1 = matrix(rep(c(0,0),2680),ncol=2,byrow=T)          # Make Finley's data.
x2 = matrix(rep(c(0,1),72),ncol=2,byrow=T)
x3 = matrix(rep(c(1,0),23),ncol=2,byrow=T)
x4 = matrix(rep(c(1,1),28),ncol=2,byrow=T)
dat = rbind(x1,x2,x3,x4)
obs = dat[,1]                                         # The observations.
pred = dat[,2]                                        # The forecasts.
ct = table(obs,pred)                                  # The contingency table.
n.trial = 1000                                        # Start of bootstrapping.
perf = numeric(n.trial)
for(trial in 1:n.trial){
samp = c(sample(1:sum(ct),sum(ct),replace=T))
ct = table(o[samp],f[samp])
perf[trial] = heidke(2,ct)
# perf[trial] = ct[2,2]/margin.table(ct,1)[2]         # Prob Of Detection.
# perf[trial] = ct[1,2]/margin.table(ct,2)[2]         # False Alarm Rate.
# perf[trial] = ct[1,2]/margin.table(ct,1)[1]         # False Alarm Ratio.
}
hist(pod,breaks=50,xlim=c(0,1))                       # The sampling distribution of perf.


################################################################
# This code computes the conditional mean of observation, given prediction.
# It produces the results in terms of boxplots summarizing the sampling
# distribution of the conditional mean of the observations. The observations
# and predictions are in a file called "tempr.dat", the first few lines of
# which look like this:
# 43.9 25.0
# 37.9 27.0
# 37.6 25.0
# 36.3 24.1
# 34.3 24.1
################################################################


rm(list=ls(all=TRUE))                                 # Clears things up.
library(fields)                                       # For bplot().

dat = read.table("tempr.dat",header=FALSE)
pred = dat[,1]
obs = dat[,2]
```

```
# bias-variance decomposition:
mse = mean ((pred-obs)^2)                          # MSE
bias = mean(pred-obs)                              # Bias
variance = var(pred-obs)                           # Note: mse = var + bias^2
pred.fix = pred - bias                             # Bias-corrected forecasts.

n.trial = 1000                                     # Number of bootstrap trials.
bin = 2.0                                          # Bin size for prediction.
n.samp = length(pred.fix)                          # Sample size per trial = pop size
xbin = seq(min(pred.fix),max(pred.fix),bin)
ybin = matrix(nrow=n.trial,ncol=length(xbin))

set.seed(1)
for(trial in 1:n.trial){                           # Start bootstrap trials.
samp = c(sample(1:n.samp,n.samp,replace=T))
u = pred.fix[samp]                                 # pred or pred.fix
v = obs[samp]
        for(i in 1:length(xbin))                   # For all xs in each bin
        ybin[trial,i] = mean(v[abs(u-xbin[i]) <= bin])   # mean of y's .
}

bplot(ybin,pos=round(xbin,digits=2), outlier=F,
        xlab="Prediction", ylab="Conditional Mean of Observation")
abline(0,1)                                        # Figure 3 .

#########################################################################
# This code computes the ROC cloud obtained from bootstrapping. The predictions
# and the target (0 or 1) are saved in a file called "gravity.dat", the 1st few
# lines of which look like this:
0.0724929 0
0.7511006 1
0.2094030 1
0.1746261 0
0.7371982 0
#########################################################################

rm(list=ls(all=TRUE))                              # Clears things up.

dat = read.table("gravity.dat",header=FALSE)
pred = dat[,1]                                     # Predictions,
obs = dat[,2]                                      # and observations.

n.trial = 100                                      # Number of bootstrap trials.
pod = matrix(nrow=100,ncol=n.trial)
far = matrix(nrow=100,ncol=n.trial)
for(trial in 1:n.trial){                           # Start bootstrap.
samp = c(sample(1:length(pred),length(pred),replace=T))
        for(i in 1:99){                            # Vary threshold to make ROC curve.
        thresh = i/100
```

```
        class = 0.5*(1+sign(pred-thresh))          # class=0 if pred < thresh. Etc.
        ct = table(obs[samp],class[samp])          # Contingency table at each threshold.
                if(dim(ct)[2]==2){                 # ROC only for 2 × 2 contingency tables.
                c00 = ct[1,1]; c01 = ct[1,2];
                c10 = ct[2,1]; c11 = ct[2,2];
                pod[i,trial] = c11/(c10+c11)
                far[i,trial] = c01/(c00+c01)
                }
        }
}

plot(far,pod,type="p",cex=0.2,xlim=c(0,1),ylim=c(0,1),xlab="FAR",ylab="POD")
abline(0,1)



#######################################################################
# This code computes the reliability diagram boxplot obtained from bootstrapping.
# The predictions and the target (0 or 1) are saved in a file called
# "gravity.dat", the 1st few lines of which look like this:
0.0724929 0
0.7511006 1
0.2094030 1
0.1746261 0
0.7371982 0
#######################################################################



rm(list=ls(all=TRUE))                          # Clears things up.
library(fields)                                # For bplot().

dat = read.table("gravity.dat",header=FALSE)
pred = dat[,1]                                 # Predictions,
obs = dat[,2]                                  # and observations.

n.trial = 100                                  # Number of bootstrap trials.
bin = 0.05                                     # 0.05 gives good-looking results.
n.samp = length(pred)                          # Sample size per trial = pop size.

xbin = seq(min(pred),max(pred),bin)
ybin = matrix(nrow=n.trial,ncol=length(xbin))

for(trial in 1:n.trial){
samp = c(sample(1:n.samp,n.samp,replace=T))
u = pred[samp]
v = obs[samp]
        for(i in 1:length(xbin))               # For all xs in each bin
        ybin[trial,i] = mean(v[abs(u-xbin[i]) <= bin])   # mean of y's .
}
```

```
bplot(ybin,pos=round(xbin,digits=2),
        xlab="Forecast Probability",ylab="Observed Fraction")
abline(0,1)


################################################################
# This code generates a discrimination plot, as well as the boxplot of the
# difference |p(y|x = 0) − p(y|x = 1)| obtained from bootstrapping.
# The predictions and the target (0 or 1) are saved in a file called
# "gravity.dat", the 1st few lines of which look like this:
0.0724929 0
0.7511006 1
0.2094030 1
0.1746261 0
################################################################


rm(list=ls(all=TRUE))                                  # Clears things up.
library(fields)                                        # For bplot().
library(verification)                                  # For discrimination.plot().

dat = read.table("gravity.dat",header=FALSE)
pred = dat[,1]                                         # Predictions,
obs = dat[,2]                                          # and observations.

discrimination.plot(obs, pred)                         # A single discrimination plot.

n.trial = 500                                          # Number of bootstrap trials.
bin = 0.05                                             # 0.05 makes nice-looking plots.
n.samp = length(pred)                                  # Sample size per trial = pop size.

n0 = length(pred[obs==0])                              # Number of class=0 cases in data.
n1 = length(pred[obs==1])                              # Ibid for class=1.
xbin = seq(0,1-bin,bin)
nperbin0 = matrix(nrow=n.trial,ncol=length(xbin))      # N(y—x=0).
nperbin1 = matrix(nrow=n.trial,ncol=length(xbin))      # N(y—x=1).

for(trial in 1:n.trial){                               # Start bootstrap trial.
samp = c(sample(1:n.samp,n.samp,replace=T))
u = pred[samp]
v = obs[samp]
        for(i in 1:length(xbin)){
        nperbin0[trial,i] = length( u[ u-xbin[i] < bin & u-xbin[i]>=0 & v==0 ])
        nperbin1[trial,i] = length( u[ u-xbin[i] < bin & u-xbin[i]>=0 & v==1 ])
        }
}
xpos = c(round(xbin+bin/2,digits=3),round(xbin+bin/2+0.01,digits=3))
bplot(abs((nperbin1/n1)-(nperbin0/n0)),pos=xpos,
                xlab="Forecast Probability",ylab="Difference",width=0.01)
```

Acknowledgements Tilmann Gneiting and Barbara Brown are acknowledged for enlightening conversations.

References

Baldwin, M. E., Lakshmivarahan, S., & Kain, J. S. (2002). *Development of an "events-oriented" approach to forecast verification*. 15th Conference, Numerical Weather Prediction, San Antonio, TC, August 12–16, 2002. Available at http://www.nssl.noaa.gov/mag/pubs/nwp15verf.pdf.

Brown, B. G., Bullock, R., Davis, C. A., Gotway, J. H., Chapman, M., Takacs, A., Gilleland, E., Mahoney, J. L., & Manning, K. (2004). *New verification approaches for convective weather forecasts*. Preprints, 11th Conference on Aviation, Range, and Aerospace, Hyannis, MA, October 3–8.

Casati, B., Ross, G., & Stephenson, D. B. (2004). A new intensity-scale approach for the verification of spatial precipitation forecasts. *Meteorological Applications*, *11*, 141–154.

Devore, J., & Farnum, N. (2005). *Applied statistics for engineers and scientists*. Belmont, CA: Thomson Learning.

Doswell, C. A., III, Davies-Jones, R., & Keller, D. (1990). On summary measures of skill in rare event forecasting based on contingency tables. *Weather and Forecasting*, *5*, 576–585.

Du, J., & Mullen, S. L. (2000). Removal of distortion error from an ensemble forecast. *Monthly Weather Review*, *128*, 3347–3351.

Ebert, E. E., & McBride, J. L. (2000). Verification of precipitation in weather systems: Determination of systematic errors. *Journal of Hydrology*, *239*, 179–202.

Efron, B., & Tibshirani, R. J. (1993). *An introduction to the bootstrap.* London: Chapman & Hall.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*, 861–874.

Ferro, C. (2007). Comparing probabilistic forecasting systems with the Brier score. *Weather and Forecasting*, *22*, 1076–1088.

Gandin, L. S., & Murphy, A. (1992). Equitable skill scores for categorical forecasts. *Monthly Weather Review*, *120*, 361–370.

Gerrity, J. P. Jr. (1992). A note on Gandin and Murphy's equitable skill score. *Monthly Weather Review*, *120*, 2707–2712.

Glahn, H. R., Lowry, D. A. (1972). The use of Model Output Statistics (MOS) in objective weather forecasting. *Journal of Applied Meteorology*, *11*, 1203–1211.

Gneiting, T., & Raftery, A. E. (2005). Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, *102*, (477), 359–378.

Gneiting, T., Raftery, A. E., Westveld, A. H., & Goldman, T. (2005). Calibrated probabilistic forecasting using ensemble model output statistics and minimum CRPS estimation. *Monthly Weather Review*, *133*, 1098–1118.

Gneiting, T., Balabdaoui, F., Raftery, A. E. (2007). Probabilistic forecasts, calibration and sharpness. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *69*(2), 243–268.

Good, P. I. (2005a). *Permutation, parametric and bootstrap tests of hypotheses* (3rd ed.). New York: Springer. ISBN 0-387-98898-X.

Good, P. I. (2005b). *Introduction to statistics through resampling methods and R/S-PLUS*. New Jersey, Canada: Wiley. ISBN 0-471-71575-1.

Hamill, T. M. (1997). Reliability diagrams for multicategory probabilistic forecasts. *Weather and Forecasting*, *12*, 736–741.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning*. Springer Series in Statistics. New York: Springer.

Heidke, P. (1926). Berechnung des Erfolges und der Gute der Windstarkevorhersagen im Sturmwarnungsdienst. *Geogra Ann. 8*, 301–349.

Jolliffe, I. T. (2007). Uncertainty and inference for verification measures. *Weather and Forecasting*, *22*, 633–646.

Jolliffe, I. T., & Stephenson, D. B. (2003). Forecast verification: A practitioner's guide in atmospheric science. Chichester Wiley.

Livezey in Jolliffe, I. T., & Stephenson, D. B. (2003). *Forecast verification*: *A practitioner's guide in atmospheric science*. West Sussex, England: Wiley. See chapter 4 concerning categorical events, written by R. E. Livezey.

Macskassy, S. A., Provost, F. (2004). *Confidence bands for ROC curves: Methods and an empirical study*. First workshop on ROC analysis in AI, ECAI-2004, Spain.

Marzban, C. (1998). Scalar measures of performance in rare-event situations. *Weather and Forecasting*, *13*, 753–763.

Marzban, C. (2004). The ROC curve and the area under it as a performance measure. *Weather and Forecasting*, *19*(6), 1106–1114.

Marzban, C., & Lakshmanan, V. (1999). On the uniqueness of Gandin and Murphy's equitable performance measures. *Monthly Weather Review*, *127*(6), 1134–1136.

Marzban, C., Sandgathe, S. (2006). Cluster analysis for verification of precipitation fields. *Weather and Forecasting*, *21*(5), 824–838.

Marzban, C., & Sandgathe, S. (2008). Cluster analysis for object-oriented verification of fields: A variation. *Monthly Weather Review*, *136*, 1013–1025.

Marzban, C., & Stumpf, G. J. (1998). A neural network for damaging wind prediction. *Weather and Forecasting*, *13*, 151–163.

Marzban, C., & Witt, A. (2001). A Bayesian neural network for hail size prediction. *Weather and Forecasting*, *16*(5), 600–610.

Marzban, C., Sandgathe, S., & Lyons, H. (2008). An object-oriented verification of three NWP model formulations via cluster analysis: An objective and a subjective analysis. *Monthly Weather Review*, *136*, 3392–3407.

Murphy, A. H. (1991). Forecast verification: Its complexity and dimensionality. *Monthly Weather Review*, *119*, 1590–1601.

Murphy, A. H. (1993). What is a good forecast? An essay on the nature of goodness in weather forecasting. *Weather and Forecasting*, *8*, 281–293.

Murphy, A. H., & Epstein, E. S. (1967). A note on probabilistic forecasts and "hedging". *Journal of Applied Meteorology*, *6*, 1002–1004.

Murphy, A. H., & Winkler, R. L. (1987). A general framework for forecast verification. *Monthly Weather Review*, *115*, 1330–1338.

Murphy, A. H., & Winkler, R. L. (1992). Diagnostic verification of probability forecasts. *International Journal of Forecasting*, *7*, 435–455.

Nachamkin, J. E. (2004). Mesoscale verification using meteorological composites. *Monthly Weather Review*, *132*, 941–955.

Raftery, A. E., Gneiting, T., Balabdaoui, F., & Polakowski, M. (2005). Using Bayesian model averaging to calibrate forecast ensembles. *Monthly Weather Review*, *133*, 1155–1174.

Richardson, D. S. (2000). Skill and relative economic value of the ECMWF ensemble prediction system. *Quarterly Journal of the Royal Meteorological Society*, *126*, 649–667.

Roebber, P. J., & Bosart, L. F. (1996). The complex relationship between forecast skill and forecast value: A real-world analysis. *Weather and Forecasting*, *11*, 544–559.

Roulston, M. S., & Smith, L. A. (2002). Evaluating probabilistic forecasts using information theory. *Monthly Weather Review*, *130*, 1653–1660.

Seaman, R., Mason, I., & Woodcock, F. (1996). Confidence intervals for some performance measures of Yes-No forecasts. *Australian Meteorological Magazine*, *45*, 49–53.

Stephenson, D. B., Casati, B., & Wilson, C. (2004). *Verification of rare extreme events*. WMO verification workshop, Montreal, September 13–17.

Venugopal, V., Basu, S., & Foufoula-Georgiou, E. (2005). A new metric for comparing precipitation patterns with an application to ensemble forecasts. *Journal of Geophysical Research*, *110*, D8, D08111 DOI: 10.1029/2004JD005395.

Wilks, D. S. (1995). *Statistical methods in the atmospheric sciences* (467 pp.). San Diego, CA: Academic Press.

Wilks, D. S. (2001). A skill score based on economic value for probability forecasts. *Meteorological Applications*, *8*, 209–219.

Wilson, L. J., Burrows, W. R., & Lanzinger, A. (1999). A strategy for verification of weather element forecasts from an ensemble prediction system. *Monthly Weather Review*, *127*, 956–970.

Decision Trees

4

G. R. Dattatreya

4.1 Introduction

Statistical decision-making is widely used in experimental earth sciences. The topic plays an even more important role in Environmental Sciences due to the time varying nature of a system under observation and the possible necessity to take corrective actions. A set of possible corrective actions is usually available in a decision-making situation. Such a set is also known as the set of decisions. A number of observations of physical attributes (or variables) would also be potentially available. It is desirable for the corrective action selected in a situation to minimize the damage or cost, or maximize the benefit. Considering that a cost is a negative benefit, scientists and practitioners develop a composite single criterion that should be minimized, for a given decision-making problem. A best decision, one that minimizes the composite cost criterion, is also known as an optimal decision.

The process of obtaining or collecting the values that the physical variables take in an event is also known by other names such as extracting features (or feature variables) and making measurements of the variables. The variables are also called by other names such as features, feature variables, and measurements. Among the many possible physical variables that might influence the decision, collecting some of them may pose challenges. There may be a cost, risk, or some other penalty associated with the process of collecting some of these variables. In some other cases, the time delay in obtaining the measurements may also add to the cost of decision-making. This may take the form of certain losses because a corrective action could not be implemented earlier due to the time delay in the measurement process. These costs should be included in the overall cost criterion. Therefore, the process of decision-making may also involve deciding whether or not to collect some of the measurements.

A mathematical space of the entire set of variations in the variables and their costs can be imagined in such a decision-making situation. Associated with every combination of values of variables, the overall cost of assigning a decision, including any measurement costs, can be imagined. Following this, the optimal decision for each combination of feature measurements can also be imagined. Such a mathematical representation of inter-relationships between all the variables involved is known as a "model." The variables of feature measurements, the costs, the parameters used for combining the costs to a single criterion, and every other mathematical quantity and function used in the representation of inter-relationships are relevant aspects of the model.

Unfortunately, a precise mathematical space of costs of decisions and hence the map of optimal decisions is merely hypothetical or ideal. Usually, there are uncertainties in exactly quantifying the mathematical inter-relationships required for such a construction. Some of the relationships may be deterministic. Some others may be statistical. There may be limited *a priori* knowledge to precisely quantify the statistical relationships. Finally, even with an imagined perfect mathematical space of inter-relationships, their representation and evaluation of optimal decisions may require formidable amounts of computer memory space and computations. Artificial Intelligence approaches for

G. R. Dattatreya (✉)
Department of Computer Science, University of Texas at Dallas, Richardson Texas 75083-0688, USA
Phone: 972-883-2189; fax: 972-883-2349;
email: datta@utdallas.edu

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

modeling and decision-making are helpful in many such situations. They are useful in reducing the complexity of representations. In some cases, they dynamically develop the representations of the model through the course of decision-making, instead of attempting to build a possibly unmanageably large static representation. They are also useful for approximate representation of imprecise relationships. Finally, they are useful in reducing the complexity of the computation required to evaluate optimal decisions, through the use of heuristics to evaluate nearly optimal decisions. Decision Trees is one of the Artificial Intelligence approaches and is the subject of the present chapter.

The purpose of working with a model is to help us in decision-making. In qualifying models, clarifications between various adjectives such as exact, precise, complete, and statistical are in order. A complete model accounts for all possible inter-relationships. A precise model specifies the inter-relationships without ambiguity. For example, the statement "high ambient Ozone levels cause considerable discomfort for people with respiratory sensitivity" specifies a relationship. But it is not mathematically precise due to the subjectivity of words such as "high," and "considerable." A specification may be precise, but only approximate, as opposed to being exact. Some relationships may be statistical, as opposed to being deterministic. Statistical relationships with complete, precise, and correct specifications are as good as similarly specified deterministic relationships in the following sense. In the case of statistical relationships, the statistical mean or the expected overall cost of the decision is minimized to obtain an optimal decision, as opposed to minimizing the exact overall cost.

Clearly, from the above arguments, a complete and exact model cannot usually be constructed in many Environmental Sciences applications. Even if a practitioner is willing to accept approximate but completely specified models, they may not be available in a timely fashion for many applications. Models may be only partially specified or the parameters of the model may not be accurate if observations are made and decisions are required to be made with limited resources; time is one of the resources. Meteorology is such an application. Meteorological phenomena are observable weather events. These are influenced by temperature, pressure, and water vapor, among others. These physical variables interact with one another. Their variations over the three dimensional space and time are also physical attributes and these contribute significantly to the occurrence of a meteorological event. Furthermore, the above physical attributes are very useful in predicting future meteorological events, within reasonable time frames. Although tremendous advances in research have increased the accuracy of forecasting, there is always room for improvement. Determination of ranges of various physical attributes and their combinations for accurate identifications of various important events is extremely useful. There is virtually no limit to the number of various transformations of variables and combinations of transformations that may potentially increase the accuracy of such classification. Moreover, different transformations over different ranges of attributes (and their combinations) may be required. Therefore, research on this topic is open-ended.

The present chapter studies a class of approaches for classification (decision-making) algorithms. These methods integrate models based on partial information about logical inter-relationships with statistical representations. The overall objective is to develop guided decision-making algorithms called Decision Trees. The approach is applicable in many experimental areas of earth sciences for reasons mentioned above. The final algorithm in this class is also known by other names such as *Multistage Classification* and *Hierarchical Classification*.

4.2 Decision-Making and Pattern Classification

4.2.1 Statistical Pattern Classification

In its simplest form, pattern classification (Duda et al. 2001) requires that a given data vector $\boldsymbol{x}$ be assigned to one of several known categories, $\omega_1, \cdots, \omega_k$. The data vector variable $\boldsymbol{x}$ is composed of m measurements so that

$$\boldsymbol{x} = [x(1),\ x(2), \cdots,\ x(m)]. \tag{4.1}$$

As mentioned earlier, each measurement is also called a feature, whose value is influenced by the pattern class corresponding to the data vector $\boldsymbol{x}$. Each feature may be cardinal, ordinal, or nominal valued. A cardinal valued variable takes values over continuous segments of a real line. An ordinal valued variable, over a countable

set of ordered values, such as integers. A nominal valued variable takes values from a finite set in which the values in the set have no natural order. An example of a nominal variable is the presence or absence of a phenomenon, such as the presence or absence of a particular pollutant in a material sample.

In many completely designed classification applications, we know the *a priori* class probabilities, P_i respectively, for each class ω_i. We also know the *class conditional probability density functions*, $p(x|\omega_i)$, for each class ω_i and for all vector points $\{\boldsymbol{x}\}$ in the observation space. We then maximize the *a posteriori* probability of the class to be assigned to the observed data. That is, assign class ω_i to the observed data vector $\boldsymbol{x}$ if the *a posteriori* probability of class ω_i,

$$P[\omega_i|\boldsymbol{x}] \geq P[\omega_j|\boldsymbol{x}], \text{ for every } j \in \{1, 2, \cdots, k\}. \tag{4.2}$$

Using the Bayes theorem in probability theory, an *a posteriori* class probability can be expressed as a function of its *a priori* class probability and class conditional density functions, as follows.

$$P[\omega_j|\boldsymbol{x}] = \frac{p(\boldsymbol{x}|\omega_j)P_j}{\sum_{l=1}^{k} p(\boldsymbol{x}|\omega_l)P_l}, \text{ for every } j \in \{1, 2, \cdots, k\}. \tag{4.3}$$

The denominator in the right hand side of the above equation is independent of j. Therefore, the decision rule in equation (4.2) simplifies to maximizing the numerator of the right hand side of equation (4.3) over all j. That is, assign class ω_i to the observed data vector $\boldsymbol{x}$ if

$$p(\boldsymbol{x}|\omega_i)P_k \geq p(\boldsymbol{x}|\omega_j)P_j, \text{ for every } j \in \{1, 2, \cdots, k\}. \tag{4.4}$$

The above approach to decision-making depends on the ability to statistically represent all the variations of the data, over the entire multivariate space of all the measurements.

4.2.2 Use of Logical Inter-relationships

Purely statistical approaches constitute one extreme way to model data for decision-making. At the other extreme is a set of pure logical inter-relationships. Such logical inter-relationships may be constructed through various types of data analyses other than with pure statistical models. These inter-relationships may be completely deterministic or approximated to be deterministic. In practice, a combination of logical inter-relationships and statistical data analysis is commonly employed. Logical inter-relationships can be considered to be *perfect* if its use is guaranteed to be error-free in every instance that it is used to make decisions. The same information can be considered to be *complete* if its application is guaranteed to lead to a final decision (as opposed to a partial decision) for every combination of measurements. An availability of such complete and perfect logical inter-relationships obviates statistical approaches. Such an ideal case is seldom found in applications. In real life applications, we usually have only imperfect and incomplete information about the model. This is also usually accompanied by past cases of data and any decisions that may have been made using such data. Such data are called training pattern samples. Practical decision-making algorithms are best designed with the help of both available information about logical inter-relationships and statistical training data. The following illustrates this approach with the help of simple fictitious example.

A patient visits his family-practice physician with flu-like symptoms. The cause may be upper respiratory allergies or a viral infection. Although there is no cure for viral infection, a secondary bacterial infection may follow in either case, under some conditions. Patients with a history of such risks should be treated differently in comparison with those having no such history. A possible model of logical inter-relationships is shown in Fig. 4.1.

The physician examines if the patient has fever. For three possible levels (or grades) of fever, the courses

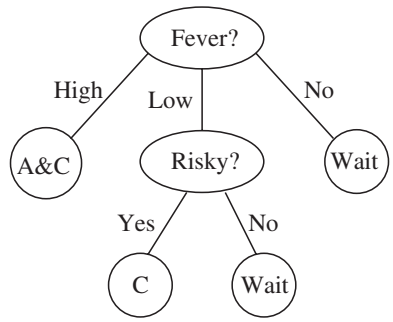


Fig. 4.1 A simple example illustrating model and statistical training

of actions are different. For low fever, the doctor examines the patient's records to check if he has risk factors. If the patient is at risk, the doctor prescribes medicines to relieve symptoms of Cold (indicated by a decision C in the figure). If the patient's fever is high, the doctor prescribes both antibiotics and Cold medicines (indicated by A & C). Under other conditions, the physician does not prescribe any medication. Of course, if the symptoms worsen within a day or two, the patient will return to the doctor's office. This is indicated by the decision "Wait." This is an example of a model of logical inter-relationships. This example assumes that the physician has a list of risk factors and there is no ambiguity about risk factors. However, this model is still imperfect since there is no specification of how to distinguish between low and high grades of fever. The final decision-making algorithm requires a threshold of body temperature to decide between low and high fever. A good threshold can be determined with the help of numerous past cases of how patients' conditions changed starting from different observed temperatures. The threshold determination may also be influenced by how past patients with different temperatures responded to different treatments. The observations about the past patients constitute statistical training data. In the above example, the physician arrives at a final decision through a sequence of partial decisions. At each stage, some information about the case (the patient in the above example) is examined and further actions are contemplated. At each stage, one of the possible actions is selected. Such an approach to decision-making is called the decision tree approach. The corresponding pictorial representation of the decision-making scheme is called the decision tree.

In a general decision-making scheme (including in decision trees), there is an optimal decision associated with every combination of feature measurements. Thus, the mathematical space of measurements is divided into regions of different optimal decisions. These regions are called decision regions. The boundaries between adjacent regions of different decisions are called decision boundaries.

4.3 Decision Regions

Decision-making algorithms induce decision boundaries and decision regions in the data space $\{\boldsymbol{x}\}$, as noted in the above section. That is, the multidimensional data space is divided into many regions and a class label is assigned to each region. There may be multiple disjoint regions constituting a single class. The following is a hypothetical example. Figure 4.2 shows an example of decision regions with four classes and two measurements, x and y.

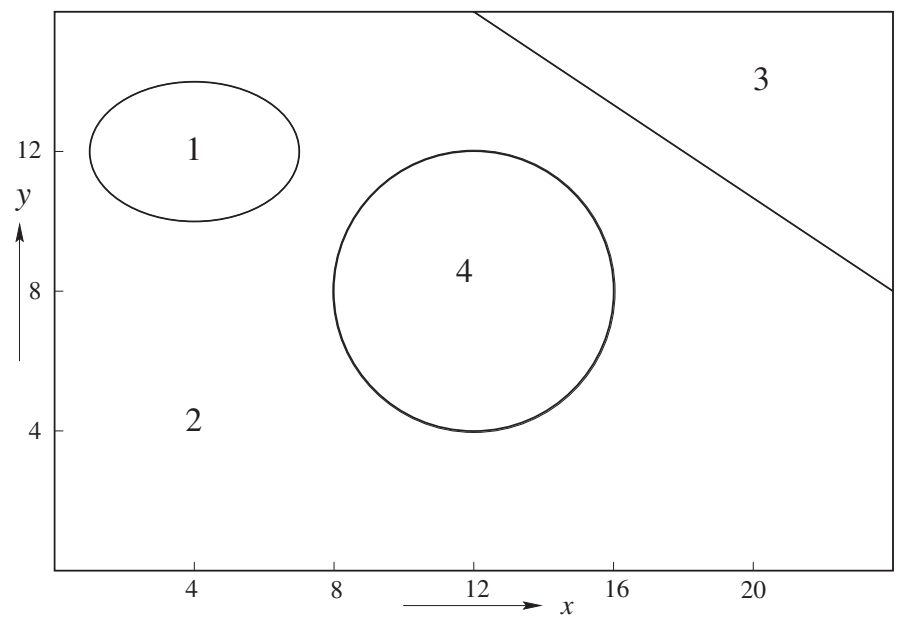


Fig. 4.2 An example of decision regions

In this example, the x axis segment extends from 0 to 24 units. The y axis segment extends from 0 to 16. The decision region for class 1 is inside an ellipse with its major axis parallel to the x axis. The ellipse is centered at (4, 12). Its major axis measures 6 and its minor axis, 4. The decision region for class 4 is inside a circle centered at (12, 8) with radius 4. The decision region for class 3 is above and to the right side of the inclined line segment passing through the two points (12, 16) and (24, 8). The rest of the area corresponds to class 2.

If the data vector also has ordinal and/or nominal features, the space of the feature measurements will be composed of continuous as well as discrete variables. In practice, the *a priori* class probabilities and the class conditional probability density functions of the observations are not accurately known. Design of pattern classifiers are then based on finite sample training data sets. There are several approaches to designing classifiers starting from training data sets. Typically, we may have access to several data vectors from each class. A labeled sample set of training data consists of the set of classes is $\{\omega_1, \cdots, \omega_k\}$ with k classes, and n_i data vectors from class ω_i where

$$\boldsymbol{x}_{ij}, \quad j = 1, \cdots, n_i \text{ and } i = 1, \cdots, k \tag{4.5}$$

is the j-th data vector from class ω_i. In some applications, the relative numbers of training data samples may adequately represent the *a priori* class probabilities. That is,

$$\frac{n_i}{\sum_{j=1}^{k} n_j} \tag{4.6}$$

may be a good estimate for P_i. In other applications, there may be attempts to supply a training data set with a maximum possible size. In such a case, the relative numbers of samples from different classes may not approximate the *a priori* class probabilities. The relative proportions in field, that is in the real application, may be known or estimated by other means. For example, during a particular season, we may know that the weather for the midday (without any additional information) has the following probabilities.

$$P[\text{sunny}] = 0.75, \tag{4.7}$$

$$P[\text{rain}] = 0.15, \tag{4.8}$$

$$P[\text{cloudy}] = 0.07, \text{ and} \tag{4.9}$$

$$P[\text{snow}] = 0.03. \tag{4.10}$$

In yet other applications, it may be reasonable to assume equal *a priori* probabilities for all classes. The data sets for individual pattern classes may be used to estimate the parameters of class conditional probability functions based on a known or assumed family of probability density function. An example of a family of probability density functions is the set of Gaussian probability density functions representing a single feature measurement. Different values for the mean and variance correspond to different members of this family of probability density functions. The mean and variance of the density function are known as the parameters of the probability density function. Other families of probability density functions may have parameters other than (or in addition to) the mean and variance. For example, a uniform probability distribution of a continuous random variable is conveniently represented by the extreme (the lowest and the highest) values that the random variable can take. In this case, these extreme values are the parameters of the probability density function. The probability density function of a random variable is the derivative of the probability distribution function. The latter is also called the cumulative distribution function. A completely specified statistical model for pattern classification or decision-making uses a probability density (or distribution) function for the feature measurements for objects (or events) from each class of patterns. The *a priori* class probabilities of the occurrence of the classes in nature are also used. All the parameter values required for decision-making are required for a complete specification. This approach of employing a family of probability density functions and using estimated values for these parameters to design a pattern classifier is known as the parametric approach. The final classification algorithm assigns a class label for every data point in the space of feature measurement, following a maximum probability of correct classification rule of equation (4.4) developed in Section 4.2. The resulting map of the class assignment divides the feature space into decision regions.

4.4 Discriminant Functions

There are alternatives to the above described parametric approach. After all, the final classifier is required to specify decision regions in the data space of the

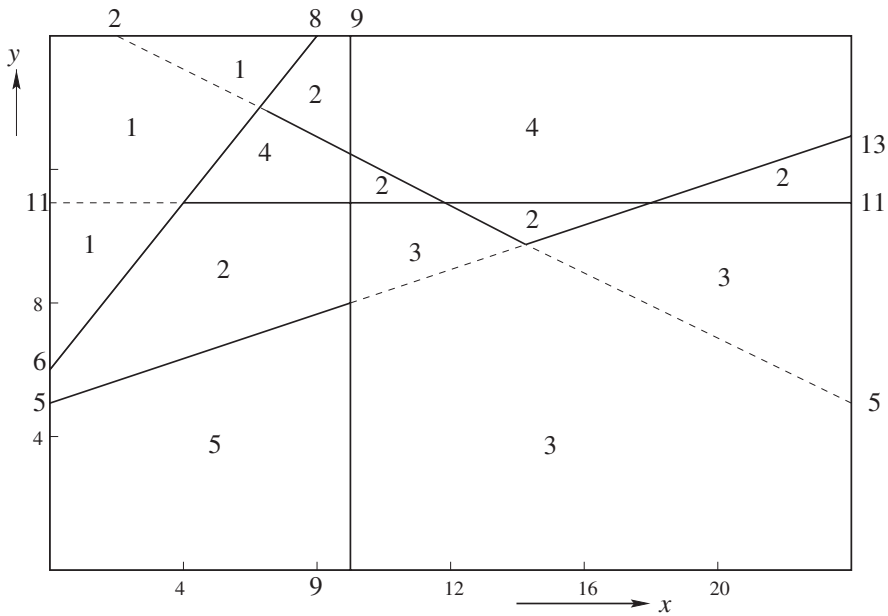


Fig. 4.3 Piecewise linear decision boundaries

available features. Of course, once the decision regions are specified, it is desirable to have an efficient algorithm to sift through the decision regions to arrive at the class label corresponding to an observed data vector. There are approaches to use parametric forms for the decision boundaries and optimize the parameters to minimize a function of the errors, based on training sets. This approach determines the parameters of such "discriminant functions." In simple cases, a discriminant function is associated with each class label. The classification algorithm evaluates these discriminant functions, for the given data, for all the classes and picks the class label that maximizes the discriminant function. In this sense, the *a posteriori* class probabilities in equation (4.2) and the joint probability density function in equation (4.4) are discriminant functions.

A classifier that uses linear discriminant functions (Duda et al. 2001) for classification is known as a linear classifier. As a simple example, consider the decision regions in Fig. 4.2. If the classification problem is to decide between class 3 and all other classes, there would be only two decision regions, on either side of the straight line separating class 3 and all other classes. The resulting classifier is a linear classifier. Linear classifiers are very popular for several reasons. The resulting decision regions are easy to visualize geometrically, in the data space. The determination of optimal parameters of linear discriminant functions are often simpler than in the nonlinear case. In some cases, original data vectors are subjected to nonlinear transformations and then optimal linear discriminant functions are developed. In the original case of full knowledge of *a priori* class probabilities and class conditional probability density functions, it can be shown that there is nothing gained by using any transformation of existing feature variables, in terms of performance of the classifier, for example in the probability of correct classification. However, in the case of design from a finite sample training data set, some nonlinear transformations may lead to a simple classification algorithm and with better accuracy of classification. A very simple generalization of linear classifiers uses *piecewise linear functions*. This is very easily illustrated with the help of another example below.

Figure 4.3 depicts decision regions for an application.

The overall rectangular region of measurements is 24 units wide and 16 high. The intercept values indicated for the line segments completely define each straight line. Every closed region has an assigned class label indicated in the figure. The broken portions of some line segments in the figure do not separate class labels. They are drawn only to clearly specify the lines. In a field application of the above pattern classifier, an observation (data) vector (x, y) would be given and the task of the classification scheme is to determine the final class label corresponding to the point (x, y) in the

decision regions. A particularly convenient approach to implement the classification algorithm for the decision regions of Fig. 4.3 is to examine which side of the different straight line segments that a given point (x, y) falls. This requires a few steps and the corresponding scheme is appropriately known as a multistage decision-making algorithm. An elegant and convenient way to represent such a classification scheme is with the help of a decision tree. Trees are a subclass of mathematical structures (objects, entities, etc.) called graphs. In many cases, such decision trees are extracted from graphs. Therefore, an elementary study of definitions and properties of graphs and trees is useful. The next section introduces principles of graphs and trees.

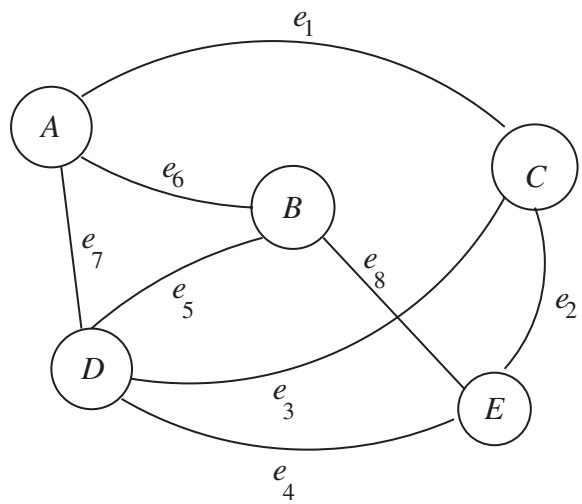


Fig. 4.4 An undirected graph

4.5 Graphs and Trees

In a graph, we have a set of vertices or nodes commonly denoted by set V. Distinct nodes are identified by numbers, letters, names, etc. Each node may represent a physical or an abstract object. An edge in a graph is an entity different from a node. An edge connects two nodes. The set of edges is commonly denoted by E. Edges are also known by other names such as arcs, branches, and links. At most one edge may be present from a vertex to another. An edge may also connect a vertex to the same vertex. The following are formal definitions.

A graph G is a set $\{V, E\}$ where V is a set of vertices and E is a set of edges defined over V. Vertices and Edges are defined as follows. A vertex is a physical or an abstract entity. An edge is a pair of vertices. A set of vertices induces a set of all possible edges. Edges can also be identified by numbers, letters, names, etc. If the same type of identifiers are used for both vertices and edges, it is important to be able to distinguish between a node identifier and an edge identifier. In the literature, it is not too uncommon to find graphs with numbers used for identifying nodes as well as edges. In such a case, we need to avoid confusion by explicitly referring to them as "vertex k" and "edge k." Pictorially, a vertex is represented by a heavy dot, a small circle, an ellipse, or a square, etc. An edge is represented by a straight or curved line drawn from one vertex to another. We use the following descriptions with obvious meanings. An edge connects its two vertices. A vertex touches its edges. An edge originates at a vertex and terminates at a vertex, etc.

Figure 4.4 depicts a graph.

Its vertex set is $V = \{A, B, C, D, E\}$ and its edge set is

$$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\}. \tag{4.11}$$

We find that $e_1 = (A, C)$, $e_2 = (C, E)$, $e_3 = (C, D)$, $e_4 = (D, E)$, $e_5 = (B, D)$, $e_6 = (A, B)$, $e_7 = (A, D)$, and $e_8 = (B, E)$. In some graphs, a direction may be associated with every edge. If an edge is not directed (also called undirected), it is considered to connect from each of its two vertices to the other. It is convenient to have all edges of a graph as directed, or all of them as undirected. This is not restrictive, since, an undirected edge can always be represented by two directed edges in opposite directions. Therefore, each pair of vertices identifying a corresponding edge in Fig. 4.4 is an unordered pair.

If all the edges of a graph are unordered pairs of vertices, the graph is said to be undirected. If all the edges of a graph are ordered pairs of vertices, the graph is said to be directed. If (R, S) is an ordered pair representing a directed edge, by convention, the edge originates at the vertex R. The number of edges originating and/or terminating at a vertex is often an important characteristic of the vertex. These are defined separately for undirected and directed graphs. In an undirected graph, the degree of a node is the number of edges that the node touches. In a directed graph, the number of edges terminating at a node is known as the

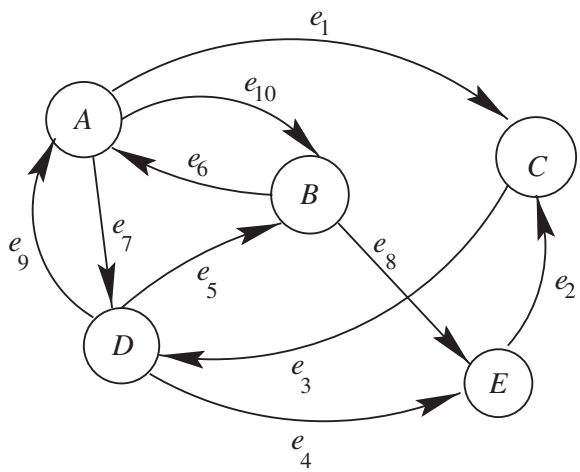


Fig. 4.5 A directed graph

in-degree of the node. The number of edges originating from a node is known as the out-degree of the node.

Figure 4.5 shows a directed graph.

It is constructed by making the following modifications to the undirected graph of Fig. 4.4. Specific directions have been imposed for edges e_1, e_2, e_3, e_4, e_5, and e_8 of the original undirected graph of Fig. 4.4. Each of the other two edges e_6 and e_7 of the original undirected graph are duplicated to create corresponding edges in opposite directions. Therefore, the vertex set for this directed graph is the same as in the above undirected graph example, given by $V = \{A,\ B,\ C,\ D,\ E\}$. The edge set E has the following ordered pairs of vertices for edges. Edge $e_1 = (A,\ C)$, $e_2 = (E,\ C)$, $e_3 = (C,\ D)$, $e_4 = (D,\ E)$, $e_5 = (D,\ B)$, $e_6 = (B,\ A)$, $e_7 = (A,\ D)$, $e_8 = (B,\ E)$, $e_9 = (D,\ A)$, and $e_{10} = (A,\ B)$. The in-degree of vertex A is 2 and its out-degree is 3. The in-degree of vertex C is 2 and its out-degree is 1. Clearly, an n vertex directed graph can have a maximum of n^2 edges. An n vertex undirected graph can have a maximum of $\frac{n(n+1)}{2}$ edges. These numbers include possible edges from a node to itself. If we exclude possible edges from a node to itself, the number of possible edges in an n-vertex directed graph is $n(n-1)$; for an undirected graph, it is $\frac{n(n-1)}{2}$.

A path, in a graph, is a sequence of $k+1$ vertices $\{v_0,\ v_1, \cdots, v_k\}$ and a corresponding sequence of k edges

$$\{(v_0,\ v_1),\ (v_1,\ v_2), \cdots, (v_{k-2},\ v_{k-1}),\ (v_{k-1},\ v_k)\}, \tag{4.12}$$

provided such edges exist in the graph to create the path $\{v_0,\ v_1, \cdots, v_k\}$. Note that v_i is a variable vertex in the above definition, in the sense that v_i is some vertex of the graph under consideration. The number of edges in a path is known as the number of hops, length of the path, etc. The first vertex in the sequence of vertices forming a path is known by different names such as the initial, originating, beginning, or the starting vertex of the path. Similarly, the last vertex in a path is known by different names such as final, terminating, ending vertex. We use descriptions such as "traversing the path," "traversing the graph through the path," etc. Some simple examples help in understanding the definition of a path. In the undirected graph of Fig. 4.4, the sequence of vertices $\{A,\ D,\ C,\ E\}$ is a path of length 3. The sequence of vertices $\{A,\ B,\ C\}$ is *not* a path since the graph has no edge between the vertices B and C. In Fig. 4.4, again, the sequence of vertices $\{C,\ A,\ B,\ D,\ A\}$ is a path of length 4 from C to A, even though the vertex A is traversed twice. In many applications, such paths are undesirable and paths without repeating vertices are desired. In the directed graph of Fig. 4.5, the sequence of vertices $\{A,\ C,\ D,\ B\}$ is a path of length 3 from node A to node B. The sequence of nodes $\{A,\ B\}$ is also a path, of length 1, from node A to node B. However, the sequence of nodes $\{A,\ C,\ E\}$ is *not* a path since there is no directed edge $(C,\ E)$ in this directed graph.

A path originating from a node and terminating at the same node is called a cycle. For example, $\{A,\ B,\ D,\ E,\ C,\ A\}$ in the undirected graph of Fig. 4.4 is a cycle of length 5. The path $\{C,\ D,\ E,\ C\}$ in the same figure is also a cycle, of length 3. In the directed graph of Fig. 4.5, each of the paths $\{A,\ B,\ A\}$ and $\{B,\ E,\ C,\ D,\ A,\ B\}$ is a cycle. An undirected graph is said to be connected if there is a path from every vertex to every other vertex. The undirected graph of Fig. 4.4 is a connected graph. A connected undirected graph without any cycle is called a tree.

We usually form a tree from a connected undirected graph by removing some edges to satisfy the requirements. Since there are no cycles in a tree, there must be one and only one path from any vertex to any other vertex. The reason such a graph is called a tree is that we can identify any node and branch out to all other nodes, just as a real tree starts from the ground and branches off to its ends. The following is a fundamental result in Graph theory.

An $n > 0$ vertex tree has exactly $n - 1$ edges. The statement is proved by induction as follows. A tree with one vertex has no other vertex to connect to and hence has no edges. A tree with two vertices must have exactly one edge connecting the two nodes. Therefore, the theorem is valid for trees with $n \leq 2$ nodes. Let every tree with a particular value of $k > 2$ nodes have exactly $k - 1$ edges. Now, add a node to such a tree to make it a $k + 1$ node graph. In order to make the new graph connected, we must add an edge between the newly added $(k + 1)$-th node and any other node, increasing the number of edges to k. When we add this new node and a corresponding new edge, we will have a path from every node to every other node, so that the graph is connected. If we now add an additional $(k + 1)$-th edge between any two nodes, say, between nodes A and B, we create a *second* path between nodes A and B, since there was already an existing path. The two paths create a cycle and hence adding the $(k + 1)$-th edge destroys the tree. Similarly, any additional number of edges over and above k edges will create one or more cycles. This shows that if every k node tree has $(k - 1)$ edges, then every $(k + 1)$ node tree has k edges. Since the property is known to be true for $k = 1$ and 2, it follows that every $n > 0$ vertex tree has exactly $n - 1$ edges.

In a rooted tree, any particular node of the tree is identified as the root node. If a node of a rooted tree is not its root and if its degree is one, we call it a leaf node or a terminal node. Figure 4.6 shows a rooted tree. It is a sub-graph of the undirected graph in Fig. 4.4. Node B is identified as the root node. Nodes C, D, and E are leaf nodes.

Starting from the root of a tree, we can reach all the nodes by the following informal traversal algorithm. Start from the root node. Traverse along any path without reversing the direction of traversal. Color or mark every traversed edge. Also mark the direction in which every edge is traversed. When a leaf node is reached, start from any non-leaf node that has been reached in an earlier traversal and that touches an unmarked edge. Continue traversing until another leaf node is reached. Continue this procedure until all edges are marked. Notice that once we start traversing from a node, we never reach a node that has already been visited, since there are no cycles in the tree. The second observation is that every edge is traversed only once. As a consequence, the above procedure assigns directions to every edge in the tree. Every direction moves away from the root node and towards leaf nodes. The third observation is that given a rooted tree, the directions associated with every edge along every path towards a leaf node is unique.

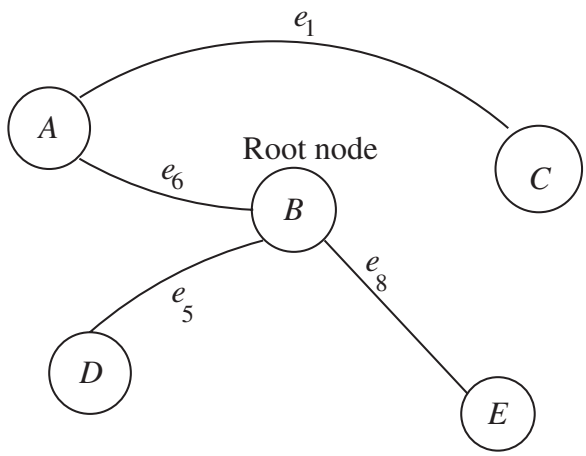


Fig. 4.6 A tree obtained from the graph of Fig. 4.4

A rooted tree in which obvious directions are assigned for every edge to create a (directed) path from the root node to every leaf node is a rooted and directed tree. Along every path to a leaf node, a non-root node is called the child of the node preceding the node in the path under consideration. All the child-nodes of a node are called siblings. If a node B is a child node of the node A, then A is called the parent node of node B. Clearly, the root node has no parent. Every leaf node has no child node. A consequence of the property that directions of edges in a rooted and directed tree are unique is that we may elect to not explicitly show the directions of edges in figures. In this sense, a rooted tree and the corresponding rooted and directed tree are one and the same. In the rooted tree of Fig. 4.6, nodes A, D, and E are siblings and all these are the child-nodes of the node B. It is convenient to redraw a rooted tree with the root node at the top and all its child nodes below the root node. Similarly, all succeeding child nodes of all newly drawn nodes are drawn below the corresponding parent nodes. Such modifications to a graph do not change a graph, since the definition of the graph (that is, the set of vertices, the set of edges over the set of vertices, directions of edges, if any, and the root node), are not changed. Figure 4.7 shows a such a redrawn tree of the one in Fig. 4.6.

The height of a rooted tree is the length of the longest path from the root node to any leaf node in

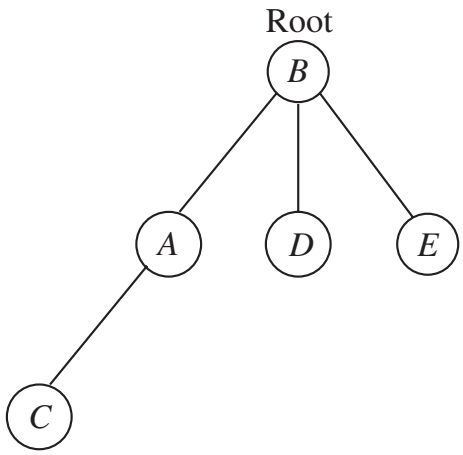


Fig. 4.7 The tree of Fig. 4.6 with a root at the top and branching downwards

the tree. The height of the rooted tree in Fig. 4.7 is two.

A directed graph that has no cycles is called a directed acyclic graph (DAG). Unlike in a tree formed from an undirected graph, it is not necessary for a DAG to have a node from which there are paths to all other nodes. Figure 4.8 is a simple example of a DAG illustrating this peculiarity.

Multiple paths between a pair of nodes are allowed in a DAG. For example, the DAG in Fig. 4.8 has two paths from node A to node E. In some decision tree design algorithms, decision trees are extracted as subgraphs of DAGs. If in a rooted tree, every non-leaf node has exactly two child nodes, the tree is called a binary tree. We are now ready to define a decision tree.

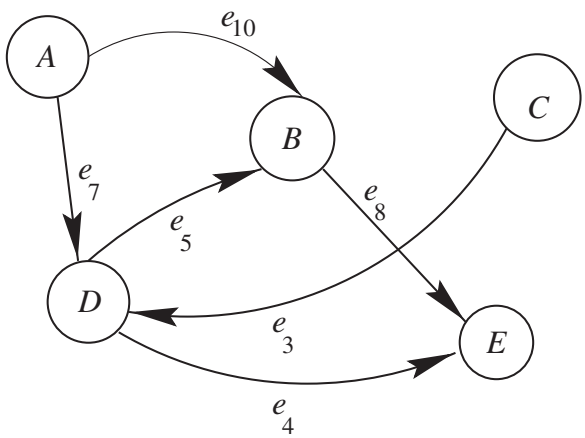


Fig. 4.8 A DAG with no unique root node, constructed as a subgraph of Fig. 4.5

Consider a rooted (and directed) tree. Let every non-leaf node examine a data vector. The data vector is specific to the non-leaf node under consideration. The data vector examined at a node can be a specific transformation of the overall data vector in the application system. Let the set of possible outcomes of the examined data vector be partitioned into as many proper subsets as the number of child nodes of the non-leaf node under consideration. Let there be a simple test at every non-leaf node to determine which child node to traverse to, at every non-leaf node. Let every leaf-node be assigned a decision from a set of decisions. Such a tree is called a decision tree. The motivation for such a definition is the following. If the data vector used at the root node is specified, the partial decision of which of its child nodes to traverse to can be determined. If the data vector at that child node is available, the next level child node to traverse to can be determined. This process can continue with partial decisions until a leaf node is reached. The decision tree is merely an approach or an algorithm to determine the final decision for every possible overall data vector point. We say that every non-leaf node "tests" a given data vector and decides the child node to which the decision-making algorithm traverses. Note that two or more different leaf nodes may be identified with the same decision. Decisions can be identified by numbers, names, subscripted symbols, etc. A decision tree extracted from a DAG has one node from which there are paths to all terminal nodes of the DAG. Such a DAG is identified by the following definition.

If a DAG has one node from which there is a path to every terminal node of the DAG, the DAG is said to be a rooted and directed acyclic graph (RDAG). Figure 4.9 shows an RDAG extracted from the directed graph of Fig. 4.5. Since the edges in an RDAG are directed, the root node is the unique node D, in Fig. 4.9.

A binary decision tree is a binary tree as well as a decision tree. Each test in a decision-making node of a binary decision tree is a test with two possible outcomes, "yes" or "no." In this chapter, we use the convention that the left branch of a decision-making node in a binary decision tree corresponds to the outcome "yes," and the right branch, to "no." Figure 4.10 is a binary decision tree; it is obtained by modifying the original (non-binary) decision tree of Fig. 4.1.

The above definitions of the decision tree and the binary decision tree are very general. It is clear that a

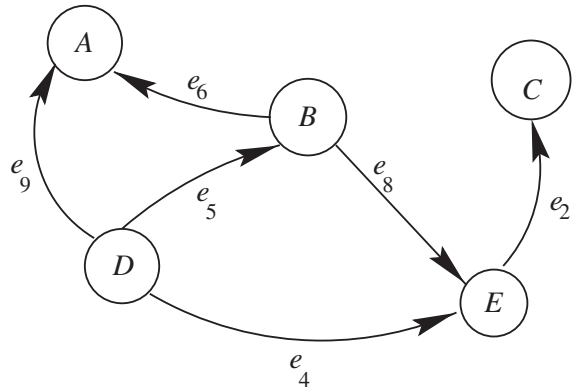


Fig. 4.9 An RDAG extracted from the directed graph of Fig. 4.5

decision tree simply specifies decision regions in the overall data space. That is, a decision tree is merely a convenient artifice to implement a decision-making algorithm over a completely specified set of decision regions. To be useful, the tests at individual non-leaf nodes must be simple. Therefore, the art of designing decision trees can actually influence the partitioning of the overall data space into decision regions, often in an interactive and iterative approach. We will study design of decision regions and decision trees in later sections; the following section illustrates implementation of decision trees for given decision regions, through examples.

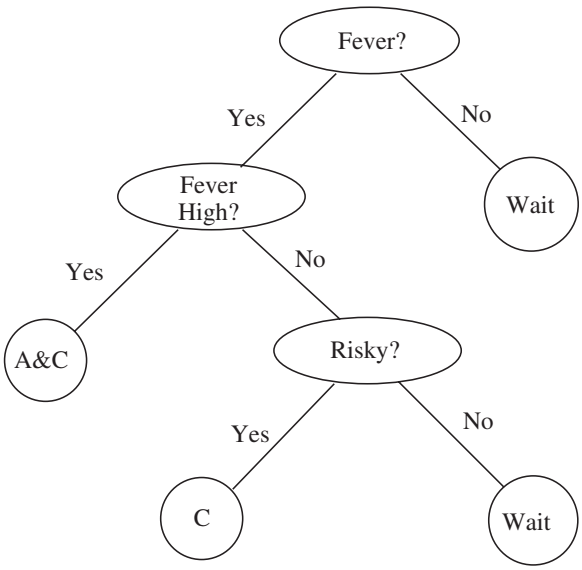


Fig. 4.10 A binary decision tree obtained by modifying the decision tree of Fig. 4.1

4.6 Decision Tree Examples

The implementation of a pattern classifier algorithm for a given set of decision regions (for example, the decision regions in Fig. 4.2 or Fig. 4.3) requires a sequence of steps. Typically, for a given data point, that is, a point (x, y), we should sequentially determine on which side of various straight or curved lines the point (x, y) falls. If we have a three dimensional data vector, the geometrical figure separating two adjacent regions is a surface. If we have a higher dimensional data vector, the mathematical equation separating two adjacent regions in the data space is referred to as a hypersurface. If we are careful, it may not always be necessary to determine the correct side of *every* line (in our two dimensional data vector cases) found in the overall region of possible measurements. In a sequence of steps, results of currently examined lines can direct us to select which line to examine next, until a final determination of the class label is made. Such algorithms are appropriately called multistage classification schemes. The distinction between a general multistage classifier and a decision tree is somewhat subjective. Generally, in decision trees, the evaluation of the test function and determination of the next action at every stage is very simple. As an example, a decision to pick one of two options based on a threshold comparison of a single variable at every stage is a very simple multistage classifier. Such a classifier is also a binary decision tree.

Figure 4.11 shows decision regions for another pattern classifier. Notice that every decision boundary is a straight line segment parallel to one or the other of the two coordinate axes. Again, the overall rectangle is 24 units wide and 16 units high, starting from the bottom left corner as the origin. The abscissa and ordinate values of the line segments, as necessary, are given outside the overall rectangle to completely specify the decision regions. The class labels of the regions are the numbers 1, 2, 3, 4, and 5, and these are given inside the decision regions.

We can implement a decision making algorithm for this classifier in the form of a decision tree. In particular, we consider a binary form of decision trees. Recall that in a binary decision tree, each decision-making node examines a condition, the result of which can be yes or no, until a final decision about the class label is unambiguous. Figure 4.12 shows such a binary

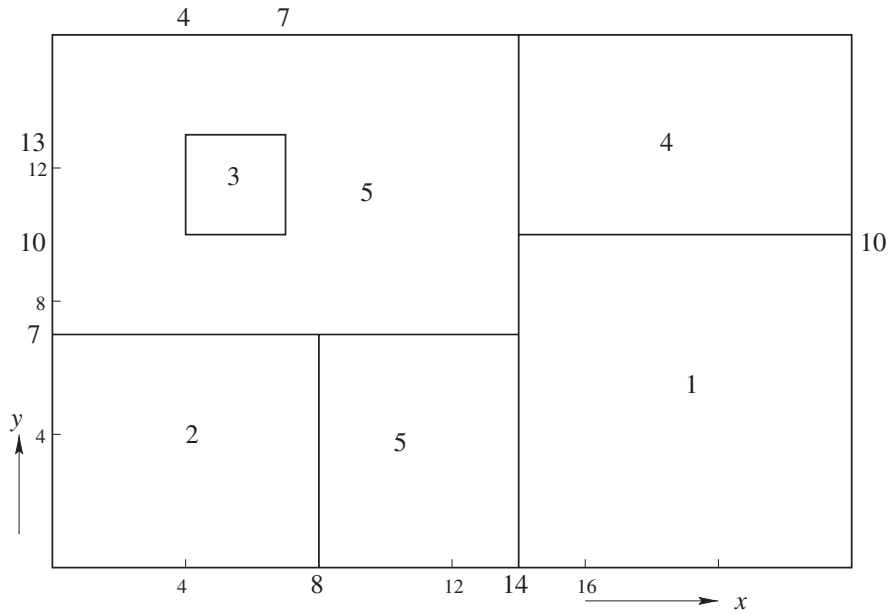


Fig. 4.11 Rectangular decision boundaries

tree in which each stage is a comparison of one of the features, x or y to a specified threshold.

The first stage examines if $y < 7$. As stated earlier, the left branch is for yes (or true), and right branch, for no (or false), a convention used here.

Clearly, we can construct such decision trees in which a test at a particular stage can produce one of many (more than two) possible outcomes. In this case, the result is *not* a binary tree but a more general tree. Figure 4.13 shows such a tree for

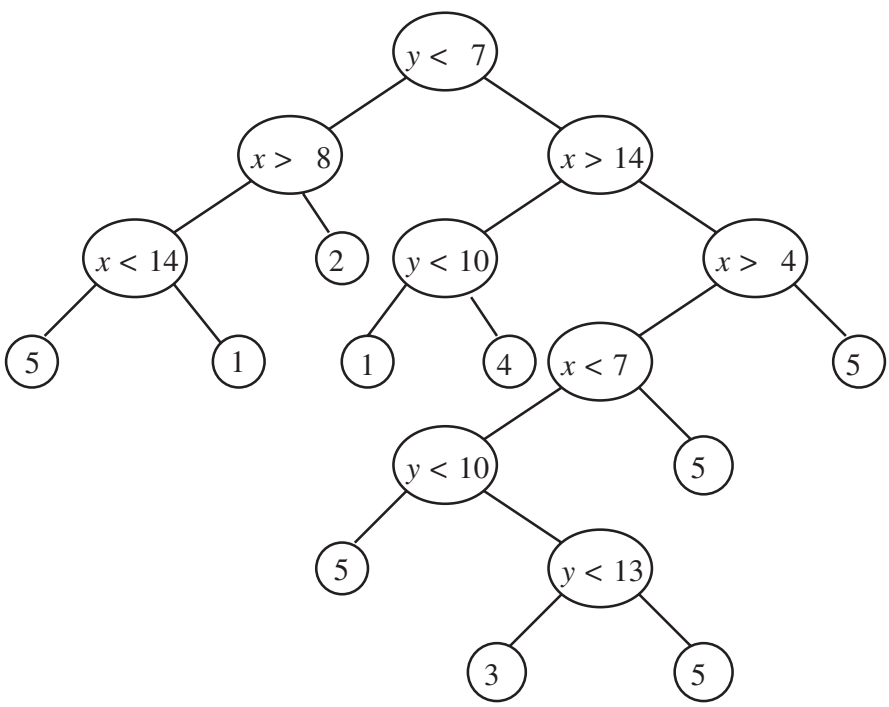


Fig. 4.12 A binary decision tree for the example of Fig. 4.11

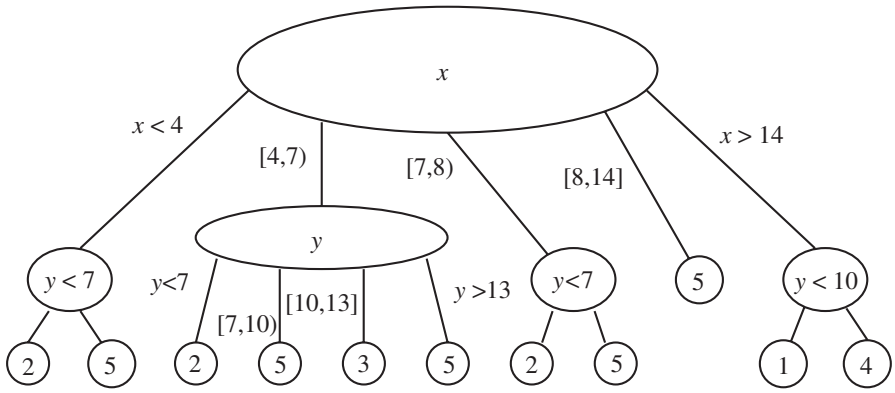


Fig. 4.13 A non-binary decision tree for the example of Fig. 4.11

implementing a decision-making algorithm for the example in Fig. 4.11.

In Fig. 4.13, if the test node indicates a condition with only two answers, the left branch is for the "true" answer and the right branch, for the "false" answer to the test. At some stages, multiple branches indicate the conditions under which a particular branch should be followed.

We can also construct decision trees in which a test can involve more than one feature variable. An appealing example is the following. The decision boundaries in Fig. 4.3 are all straight line segments. But these straight lines are *not* parallel to either the x or the y axis. However, each straight line is expressed by a simultaneous linear equation involving both the variables x and y. The two sides of one such equation is specified by the two possible outcomes of the corresponding inequality. Thus we can construct a binary tree for the decision regions of Fig. 4.3 in which each test is the examination of an inequality involving a linear combination of both the variables x and y. Constructing such a tree for the decision regions of Fig. 4.3 is a comprehensive exercise and is suggested for the reader.

The decision boundaries are not required to be straight line segments to implement the decision making algorithm as a decision tree. Of course, if tests at individual stages of a decision tree are very complicated, the use of the tree approach to decision-making is questionable. Let us implement a decision tree for classification over the decision regions of the example in Fig. 4.2. The basic equation for an ellipse whose center is the origin of the coordinate plane and whose major and minor axes are respectively collinear with the coordinate axes x and y is

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{4.13}$$

where a is the intercept on the positive x axis and b, on the positive y axis. The major axis, the longest distance between two extremities of the ellipse and passing through its origin is $2a$. The minor axis, the shortest distance, is $2b$. Shifting the center of the ellipse and substituting for the values of the semi-major and semi-minor axes for our example, we have the equation for the ellipse as

$$\frac{(x-4)^2}{9} + \frac{(y-12)^2}{4} = 1. \tag{4.14}$$

Simplifying, we obtain the equation for our ellipse as

$$4x^2 - 9y^2 - 32x - 216y + 1324 = 0. \tag{4.15}$$

The region corresponding to the inside of the ellipse is class label 1 and it is the set of all $(x,\ y)$ following the inequality

$$u = 4x^2 - 9y^2 - 32x - 216y + 1324 < 0. \tag{4.16}$$

In the above inequality (4.16), the variable u is defined to be a composite feature variable, one that is derived as a transformation from the original feature measurements of x and y.

The equation for a circle of radius r and centered at the origin is

$$x^2 + y^2 = r^2. \tag{4.17}$$

Translating to the center of the circle in our example and using the radius of 4 units, we have

$$(x-12)^2 + (y-8)^2 = 16 \tag{4.18}$$

which simplifies to

$$x^2 + y^2 - 24x - 16y + 192 = 0. \tag{4.19}$$

The region corresponding to class 4 is inside and it is the set of all $(x,\ y)$ satisfying the inequality

$$v = x^2 + y^2 - 24x - 16y + 192 < 0. \tag{4.20}$$

As in the earlier case, the variable v is a composite feature variable, derived from the original feature measurements x and y.

If a straight line passes through two points $(x_1,\ y_1)$ and $(x_2,\ y_2)$, its equation is given by

$$\frac{y - y_1}{x - x_1} = \frac{y_2 - y_1}{x_2 - x_1}. \tag{4.21}$$

The straight line delineating class 3 from other classes in our example lies on the two points (12, 16) and (24, 8). Therefore, its equation is

$$\frac{y - 16}{x - 12} = \frac{8 - 16}{24 - 12}. \tag{4.22}$$

Simplifying, the equation for the straight line is

$$2x + 3y - 72 = 0. \tag{4.23}$$

The region of class 3 is the set of all $(x,\ y)$ corresponding to the inequality

$$w = 2x + 3y - 72 > 0. \tag{4.24}$$

Again, w is a derived feature variable from the original measurements x and y. With this preparation, the decision tree implementation for classification in this example is simple. Figure 4.14 shows such a tree. The tests at individual stages in the tree are simple threshold comparison; however, the variables which are compared with the thresholds are nonlinear transformations for the ellipse and the circle and an affine transformation for the straight line.

The above examples demonstrate several characteristics of decision trees, their constructions and manipulations. All these decision trees are constructed with the help of three different examples of decision regions marked and given to us. Thus, these approaches are directly useful if any of the following is satisfied.

1. We know the decision regions.
2. We know the *a priori* class probabilities and the class conditional probability distribution functions of the measurement vector. In this case, the decision regions can be mathematically developed with the help of the maximum *a posteriori* probability decision-making scheme, given by equation (4.4).

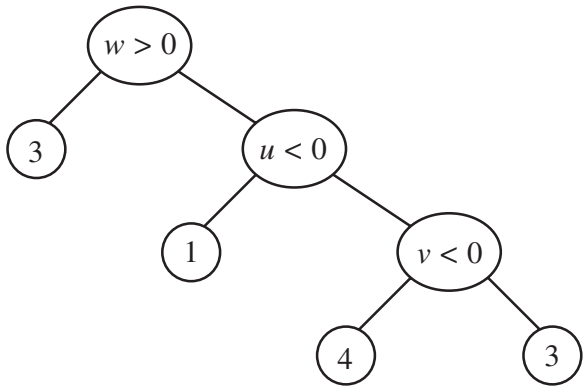


Fig. 4.14 A binary decision tree for the example of Fig. 4.2

In many practical applications, pattern classifiers are required to be designed with the help of partial knowledge about the physical system, some training data with sample sets of known class labels, unclassified data, or some combination thereof. In such cases, we can attempt to design classifiers that can be implemented as efficient decision trees. That is, instead of designing the decision boundaries and then converting them to decision trees (as is the case in earlier examples), the design of decision regions and a corresponding decision tree can proceed hand in hand. But, first, the design of decision trees from a given set of decision regions is discussed in the next section.

4.7 Decision Tree Design from Decision Boundaries

The above examples in Section 4.6 illustrate the development of decision trees from given decision regions. In this section, a systematic design methodology and possible optimization approaches are studied for the same problem. The decision boundaries may be given merely as a convenient approach to partition the space of feature measurements. But the actual assignment of a class label (decision) to each region enclosed by the the decision boundaries may not be available. In such a case, assignment of a class label to the decision regions is part of the decision tree design problem.

This section develops the concepts of the overall graph from which decision trees can be extracted, an appealing criterion for optimization, and descriptions of the general approach to optimal tree design. A simple but fairly general example of decision boundaries is used to illustrate the concepts and approaches. A detailed presentation appears in Dattatreya and Kanal (1985), which also includes a discussion on its suitability for problems with a modest number of features, each of which is quantized to only a modest number of levels.

4.7.1 The Feature Space

The decision boundaries are assumed to be hyperplane segments parallel to the coordinate axes of the m feature variables $x(1), \cdots, x(m)$. As demonstrated with examples in Section 4.6, this assumption is not restrictive since nonlinear transformations can be represented by new feature variables. Let the feature variable $x(i)$ be partitioned into n_i segments with $n_i - 1$ threshold values. Name these segments as 0, 1, $\cdots$, $n_i - 1$ for convenience. That is if $x(i)$ is in the j-th segment, we say that $x(i) = j$. The entire feature space is partitioned into $\prod_{i=1}^{m} n_i$ regions. Each region is bounded by hyperplanes parallel to the coordinate axes. Each such region is represented by a point in a lattice mathematically represented by the vector $\boldsymbol{x} = [x(1), \cdots, x(m)]$. Each point can be called a cell in the lattice. A sublattice is obtained by fixing one or more of the $x(i)$ variables to some constants. Sublattices can also be formed by successively cutting and considering one part of the lattice parallel to one or more coordinate axes.

4.7.2 Problem Formulation

The pattern classification problem is that a class label from the set of classes $\Omega = \{\omega_1, \cdots, \omega_k\}$ be assigned for every cell, with a minimum expected cost of decision-making. In the completely specified probabilistic framework, we will also be given the *a priori* class probabilities $P_1, \cdots, P_k$ respectively for the k classes, the class conditional probability values $P[\boldsymbol{x}|\omega_i]$ for each cell and for each class ω_i, and the elements c_{ij} of the cost matrix of classifying a data vector belonging to class ω_i with a decision of ω_j.

4.7.3 Optimization Criterion

Clearly, if the only cost involved in assigning a decision to a data vector (pattern sample) is the classification cost, we should determine the expected costs of assigning a cell to different classes and choose a class label for each cell based on the criterion of minimum expected cost. The expected cost of assigning a cell $\boldsymbol{x} = [x(1), \cdots, x(m)]$ to the class label ω_j is obtained as follows. Let $S(\omega_j)$ be the random variable cost of assigning class ω_j.

$$E[S(\omega_j)|x] = \sum_{i=1}^{k} c_{ij} \frac{P[x|\omega_i]P_i}{\sum_{l=1}^{k} P[x|\omega_l]P_l}. \quad (4.25)$$

However, if obtaining feature measurements involves costs, then for some cases, it may be advantageous to make a class assignment based on less than all the feature measurements. The following applications illustrate this scenario. In medical diagnosis, obtaining some test measurements may be time consuming (risking a worsening condition), costly, invasive, or risky to the body. In meteorology, numerous measurements and numerically evaluated derived features are potentially available. Some of them require fine grain modeling of the atmosphere and are computationally inefficient to calculate all the time. However, certain combinations of other routine measurements may require a more careful consideration of these expensive features, especially for decisions on severe weather patterns. Other applications are in banking and other financial institutions to determine whether or not to increase interest rates, whether or not to change prices of commodities, process control, root cause analysis (Wilson et al. 1993) of adverse event occurrences in industry, etc. In all these, decisions may normally be made based on commonly available measurements. The overall expected cost of decision-making may be less if additional, expensive measurements are extracted for certain combinations of values taken by the commonly available measurements. Let r_i be the cost of making the measurement to extract the feature $x(i)$. Of course r_i and c_{ij} are in the same physical dimensions so that the total expected cost of assigning a class label to a data vector is the sum of the feature measurement costs and the expected cost of classification, conditioned on the observed measurements. The decision tree design problem is

to develop a tree that minimizes the expected overall cost of making the decision on a pattern sample.

4.7.4 Solution Approach

The general approach to designing decision trees uses the "Principle of Optimality." If we already know the optimal decision-making algorithms for some sublattices, the principle of optimality helps us in optimizing over larger sublattices that can be formed by the union of original sublattices. Algorithms that use the principle of optimality are known as "Dynamic Programming" algorithms. The principle of optimality is stated as follows. Let a problem B be a subproblem of problem A. Consider an optimal solution to the problem A. Let the optimal solution to problem A also solve the subproblem B. In such a case, the solution of the subproblem B used by the optimal solution for the problem A is an *optimal* solution for the subproblem B. Some general conditions are required for the principle of optimality to hold for a class of problems. In our case, $r_i \geq 0$ for $i = 1, \cdots, m$ is true. Also, $c_{ij} \geq c_{ii}$, $j = 1, \cdots, k$ and $i = 1, \cdots, k$. That is, the feature measurements may incur a positive or no cost. But feature measurement costs cannot be negative. A correct classification does not incur more cost than an incorrect classification. These are sufficient conditions for the principle of optimality to hold for the present problem.

To illustrate the solution approach, consider a specific case with three feature measurements x, y, and z. Let x take values 0, 1, or 2 only. Let y take values 0, 1, 2, or 3 only. Let z take values 0 and 1 only. Let the number of classes be four. Figure 4.15 shows a directed acyclic graph of all possible feature sequences and with all their possible outcomes.

At the lowest level, every node has the measurements of all the features. Each of the lowest level nodes represents a cell. Edges from the third row of nodes to the fourth row of nodes are not drawn in Fig. 4.15 to reduce clutter. Also, all the edges in the graph are directed downwards and the tips of arrows are not drawn in the figure. The minimum expected cost of each node is the sum of the costs of all the features measured so far plus the minimum expected cost of classification. At each of the lowest level of nodes, all the feature measurements (x, y, and z) have already been obtained, incurring corresponding measurement costs. Therefore, there are no other feature measurements to consider at these lowest level nodes and a final class label decision is required to be made. Intermediate nodes represent values of measurements obtained with one or at most two of the features x, y, and z. In Fig. 4.15, if a feature is not measured, it is represented by a hyphen. Therefore, $(2, -, z)$ represents a node at which the feature x has been measured and found out to be 2, feature y is not measured, and a decision to measure z is being considered. In the Fig. 4.15, within each node, the measurements are written top-down, for lack of space. At the node $(2, -, z)$, in addition to the possibility of measuring the feature z, the option to assign a minimum classification cost class label based on the available information of $x = 2$ is possible. In order to make an optimal decision, we need to compare such minimum expected cost of classification with the overall expected cost of the alternative decision, that is of recommending making the measurement of the feature z. That is, we should already know the expected overall cost of the optimal rooted and directed acyclic

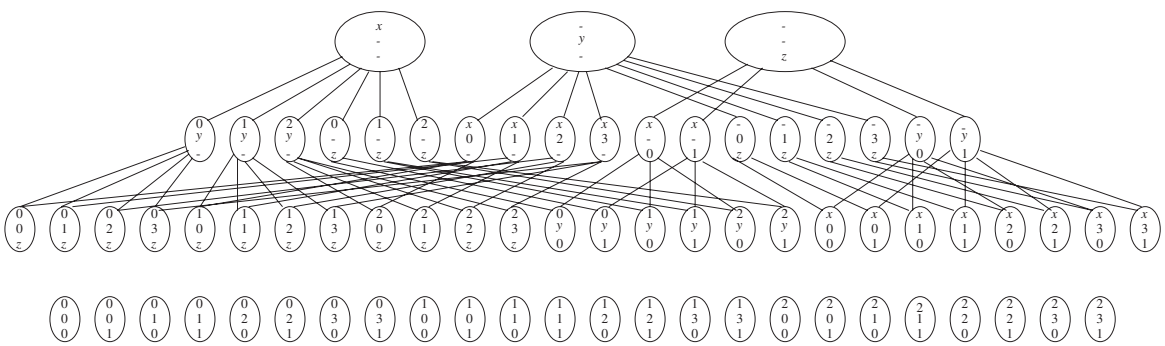


Fig. 4.15 Universal graph for optimal tree design example

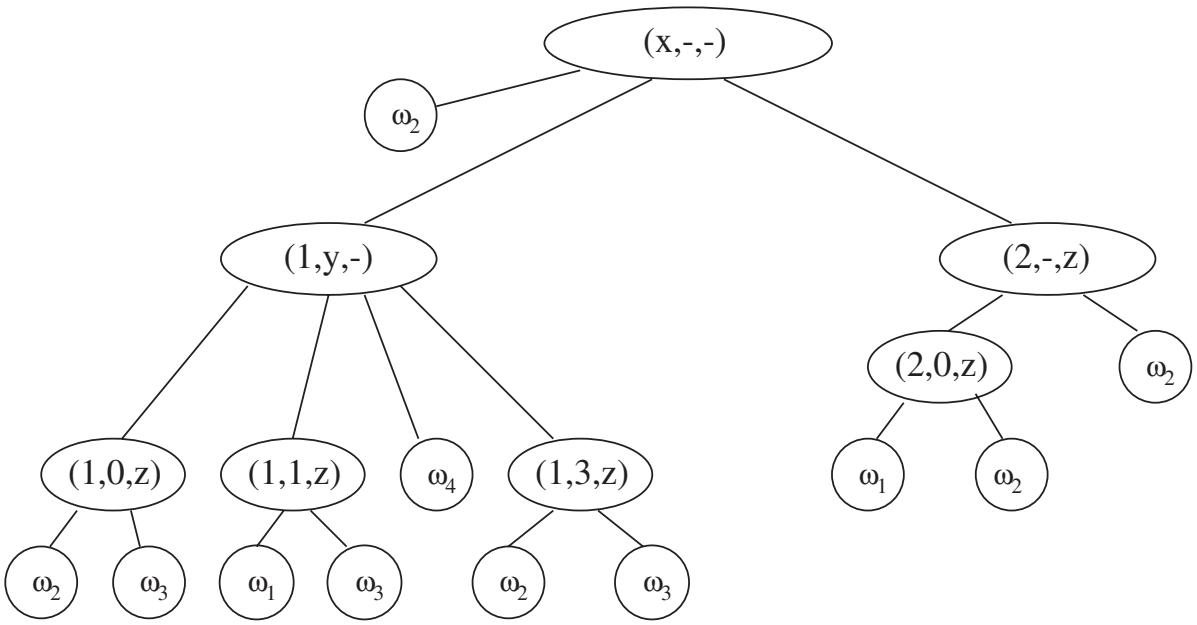


Fig. 4.16 A possible optimal decision tree in the form of a rooted acyclic graph, developed from Fig. 4.15

subgraph rooted at the node $(2, -, z)$, in Fig. 4.15. The decision tree can be designed in a bottom-up fashion, starting from assigning class label decisions to the bottom-most nodes in the DAG of Fig. 4.15. The principle of optimality plays an important role here. As a consequence of the principle, we know that if the node $(2, -, z)$ eventually becomes a part of the optimal rooted and directed acyclic graph(RDAG), then the sub-graph below the node $(2, -, z)$ must be an optimal subgraph for the intermediate problem of dealing with the partial information $(2, -, z)$. We can separate the leaf nodes of the optimal RDAG so that the final result would be a decision tree. A possible RDAG as an optimal solution extracted from the Fig. 4.15 is drawn in Fig. 4.16. In this figure, every final decision of a class label is depicted by a separate node so that the figure is a decision tree.

4.7.5 Efficient Implementation of Binary Trees from Decision Regions

In some applications requiring representation of decision tables through efficient binary trees, the class assignment for every cell is rigid and given. Such a problem of efficient implementation of decision-making through the use of binary trees can also be solved by dynamic programming procedures similar to the above approach (Payne and Meisel 1977). Many such optimal decision tree design problems are computationally complex (Murthy 1998). The fundamental reason for this is that the number of subcases to be examined increases exponentially as a function of some key parameter. Even in the case of designing a binary decision tree to examine which of the n bins a given scalar measurement falls into, the complexity is exponential in n. Therefore, many researchers recommend the use of heuristics to design decision trees. However, these comments apply more to large directory look-up problems. In practice, many pattern classification problems have only a modest number of measurements and a modest number of quantization levels. Computations for the design of optimal trees in these cases are indeed feasible.

4.8 Decision Tree Design from Labeled Training Samples

A very common starting point for the design of decision trees in application areas is with a set of labeled training samples. Simple approaches to designing

decision trees in such cases is the subject of study here. The problem of developing optimal separating hypersurfaces from the training data set is not the topic here. So far, we have seen ample evidence to the effect that the most convenient form of decision boundaries for decision tree design are segments of hyperplanes parallel to the coordinate axes of the feature space (data vector space). Hyperplane segments inclined to the coordinate axes form a second choice set. In practice, a good combination of these two types of decision boundaries serves the best. It is always possible to separate a set of labeled training samples, even from multiple classes, with the help of hyperplane segments.[1] However, the use of a decision tree is in applying the classification algorithm to new data samples. Although the new samples come from the same cause as the training samples do, the finite set training samples do not perfectly depict all the characteristics and variability in all possible samples from the original cause.

4.8.1 Linear Discriminant Functions

Decision regions separated by hyperplanes are obtained through the use of linear discriminant functions. A linear discriminant function separates the feature space into two regions separated by a hyperplane. Decision regions formed through the use of j multiple linear discriminant functions divide the feature space into a maximum of 2^j regions. It is not necessary for all of these 2^j regions to be assigned a different class label. Indeed, if the number of class labels is k, we merely need $j \geq \log_2(k)$ and $(2^j - k)$ regions in the feature space have class labels repeated from the other k regions. The general form of a linear discriminant function is

$$\sum_{j=1}^{m} w_j x(j) + w_0. \tag{4.26}$$

All the data vectors that satisfy

$$\sum_{j=1}^{m} w_j x(j) + w_0 > 0 \tag{4.27}$$

lie on one side of the corresponding hyperplane and all the data vectors satisfying

$$\sum_{j=1}^{m} w_j x(j) + w_0 < 0 \tag{4.28}$$

lie on the other side. Data vectors satisfying

$$\sum_{j=1}^{m} w_j x(j) + w_0 = 0 \tag{4.29}$$

lie exactly on the hyperplane separating the two regions. A linear discriminant function for a decision boundary that is perpendicular to the $x(j)$ axis and parallel to all other coordinate axes is of the form

$$x(j) + w_0. \tag{4.30}$$

The approach suggested here develops optimal hyperplanes between neighboring pairs of classes. This leads to decision regions separated by piecewise hyperplanes. Linear combinations of the data vectors are formed that lead to rectangular decision regions in a transformed feature space. The approaches of Section 4.7 are then applicable for completion of decision tree design. Figure 4.17 shows a simple example of data samples generated with random numbers.

Two-dimensional data samples from four classes are plotted with different symbols for points from different classes. Such plots are called scatter plots. These random numbers are generated to simulate the decision boundaries in Fig. 4.2, except for the data of class 2. Representative straight line segments are drawn for possible decision boundaries. The line segment parallel to the y axis completely separates samples from classes 2 and 3. One of the line segments also completely separates samples from classes 3 and 4. The other two line segments separate samples from corresponding pairs of classes, but these boundaries result in a few misclassified samples. It is clear that there is no single straight line that separates samples from classes 1 and 2. Similarly, It is also clear that there is no single straight line that separates samples from classes 1 and 3. It *is* possible to draw several straight line segments to completely separate the sets of samples from different categories to be in different regions. For example, the straight line parallel to the x-axis in Fig. 4.17 may be replaced by a set straight line segments to correctly classify all the given labeled training samples. Each such line would be parallel to the x-axis or the y-axis. Together, these line segments would

[1] The only exception is if two or more training samples from *different* classes are identical in all feature measurements.

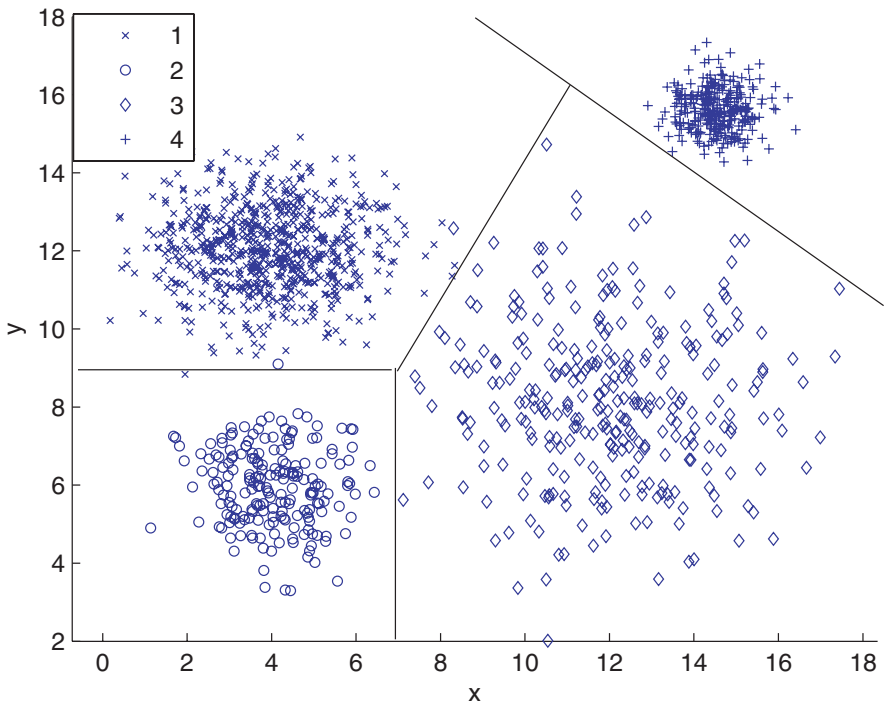


Fig. 4.17 A scatter plot example with four classes

retain all the samples belonging to class 2 on the lower side of such a complicated boundary, and all the samples belonging to class 1 on the higher side of the same boundary. However, no such approach to correctly classify all the given labeled training samples can guarantee perfect classification of future pattern samples. Such an attempt to classify all or most of labeled training samples with complicated boundaries is called "over-fitting." Such over-fitting leads to larger decision trees in which a decision region is split into multiple sub-regions with different classes. A large decision tree can be pruned to eliminate some over-fitting.

The above characteristics about decision boundaries constructed with straight lines give us clues on the types of optimizations used to find good straight lines. These are briefly described here. In the case of labeled sample sets, the distance between the mean values of samples from different classes allow us to form pairs of neighboring classes. Therefore, procedures for determining straight lines can attempt to concentrate on neighboring pairs of classes.

1. **Lines parallel to coordinate axes**
   Examining one-dimensional scatter plots in each feature dimension is very simple. The entire interval over which all the samples fall can be divided into several regions, based on whether or not these lines reasonably well separate the samples. This procedure can be applied for each individual measurement.
2. **Hyperplanes with arbitrary orientations**
   There are two subcases in this category:

   (a) In the first one, there exists a hyperplane that can completely separate samples from two classes. Such pairs of classes are called linearly separable classes. The straight line separating samples from classes 3 and 4 in Fig. 4.17 is such an example. Given two sets of training samples, one for each class label, whether or not they are linearly separable can be determined through linear programming. Intuitively, the best hyperplane to separate a linearly separable pair of classes is the one that has equal and the largest distance from the hyperplane to the nearest samples, from each class. There are mathematical programming techniques to find such an optimally separating hyperplane. The resulting hyperplane is called a Support Vector Machine. There is a rich class of other

criteria and corresponding mathematical programming algorithms to determine optimal separating hyperplanes. Duda et al. (2001), is an excellent book on this topic.

(b) In the second case, there is no hyperplane that can perfectly separate samples from two classes. Samples from classes 1 and 3 in Fig. 4.17 constitute such an example. In this case, optimization problems to minimize some meaningful criteria are formulated and solved to find optimal hyperplanes. An example of such a criterion is the minimum of the sum of squared distance from all the samples of both the classes to the hyperlane. This problem can be solved through the use of the pseudo-inverse of a matrix formed with labeled data vector samples from both the classes (Duda et al. 2001).

These are only some general suggestions to develop linear decision boundaries. The task of decision tree design from such boundaries follows approaches developed in earlier sections.

4.9 Unsupervised Decision Tree Design

In some applications, we have unlabeled or unclassified pattern samples for training. Each pattern sample is a point in the data vector space. There is no decision or class label assigned to any of the samples. Inferring the existence of any natural grouping of samples, designing a scheme to separate them into such groups, and designing a classification scheme to assign future appearances of pattern samples to one such group – all fall under unsupervised learning and classification approaches. The structure of the data set may indicate that the set naturally groups into a few categories. Each category may correspond to some physical significance. Specifically, observations within a group may point to some strongly likely consequences. In meteorology, for example, certain physical conditions may result in an unstable state or cause an imbalance. Such temporary conditions may cause further changes until the overall state stabilizes. In the multidimensional mathematical space of all physical variables, there may be several regions of stable states separated by regions of unstable states. This may be the underlying reason for the data to be found in subjectively distinct groups. Some combinations of unstable physical attributes may result in regenerative effects creating severe weather patterns. An example of physical conditions causing regenerative effects is in the occurrence of snow avalanches in mountains. A disturbance causing some movement of snow at high altitudes may trigger more disruption in the snow. Beyond a critical point, the regenerative effect can be very strong.

Another example of occurrence of natural clusters is in biological taxonomy. The overall set of plants and/or animals is hierarchically (repeatedly) split into finer and finer groups. Members in a group have strong similarities and members in different groups have strong dissimilarities. There may be cases (members) that do not clearly belong to one or another class, but such cases are rare. One reason for natural formation of such distinct hierarchical groups is that certain combinations of physical attributes favor regenerative survival.

Clustering of unlabeled data is a first step in designing a decision tree from unlabeled pattern samples. The emphasis in this chapter is not on general clustering approaches. Anderberg (1973) is a classic book on clustering. Kaufman and Rousseeuw (1990) is a more recent one on the same topic. Clustering approaches can be influenced for better design of decision trees. That is, decision boundaries parallel to the coordinate axes of the space of features are sought whenever feasible. There are two main approaches to clustering. The bottom-up or the data driven approach starts grouping data samples into pairs, groups of three, etc., and grow into larger clusters all the way to a single group. This approach is called agglomerative clustering. Some subjectively appealing quantitative criteria to determine a reasonable number of distinct clusters are available. Alternatively, top-down or model driven approaches are also available. In its most elementary form, the unlabeled training samples are divided from an initial single group all the way down to separate groups for individual data samples. This approach is known as divisive clustering. Again, appealing criteria are available to stop at a reasonable number of clusters.

Such clustering approaches finally lead to decision assignments for every sample in the entire set of pattern samples. These results should be fed to algorithms for determining decision regions. That is, instead of assigning decisions to data points in the

data space, larger regions should be identified for decision regions. Simpler approaches construct piecewise hyperplane segments parallel to the coordinate axes in the data space. Other approaches with arbitrarily oriented hyperplanes and even nonlinear hypersurfaces are possible.

In both the top-down and bottom-up methods, attempts should be made to incorporate all available information about the application system and data. Often, much of such information may be in subjective forms. The decision tree designer would use judgments and mathematical interpretations of such information. These approaches go hand-in-hand with the above approach of determining decision regions from clustered data. A particular example of such a refinement technique is "pruning of decision trees." After designing a decision tree, the designer can examine the worthiness of distinguishing some pairs or groups of decisions (either at intermediate or at final decision-making stages). Merging some decision regions with this approach eliminates some nodes in the tree and results in a pruned tree.

4.9.1 Bottom-Up Method

A pure bottom-up method of clustering successively combines (merges) sets of samples until a specified number of classes (groups) is reached. Usually, such combinations are continued until all the samples are in one single group. Thereafter, it is a little easier to decide on the number of classes and this may be influenced by a suggested number of classes. Central to determining which pair of a partition of the sample set are the *closest* for merging is the definition of a distance measure between two sets of samples. A simple distance is the Euclidean distance between the centroids of the two sample sets. If $\{\boldsymbol{x}_{1j},\ j = 1, \cdots, n_1\}$ and $\{\boldsymbol{x}_{2j},\ j = 1, \cdots, n_2\}$ are the two sets of samples, the mean data vector of the set i is

$$\boldsymbol{\mu}_i = \frac{1}{n_1} \sum_{j=1}^{n_i} \boldsymbol{x}_{ij}. \tag{4.31}$$

The Euclidean distance between two m-dimensional vectors $\boldsymbol{y} = [y(1), \cdots, y(m)]$ and $z = [z(1), \cdots, z(m)]$ is simply the geometric distance given by

$$\mathrm{d}(\boldsymbol{y},\ z) = \sqrt{\sum_{l=1}^{m} [y(l) - z(l)]^2}. \tag{4.32}$$

A drawback of the Euclidean distance is that it treats relative distances along all the feature measurement axes in the same way. In reality, different features may have different physical dimensions and may require normalization. Kaufman and Rousseeuw (1990), introduce methods for combining variables of different types into a single dissimilarity measure.

Attempting to separate the samples with hyperplanes parallel to the coordinate axes is preferable for the eventual implementation of a decision tree. This will also avoid the problem of different scales along different axes. However, during a hierarchical separation of a set of points along a line (along one feature measurement), we do not know the number of classes to split the data into. Therefore, separating or clustering over individual axes in the feature space is recommended only if sample sets are "well separated." If such an approach is viable, the resulting clusters may be further separated by the usual hierarchical approaches. A quantitative criterion of well separateness considers the spread of data samples within each class as well as the spread between the two clusters. Following is one appealing definition of such a measure. The mean square distance of all samples within a class to its centroid is a measure of spread within a class.

$$\sigma_i^2 = \frac{1}{n_i} \sum_{j=1}^{n_i} \sum_{l=1}^{m} [x_{ij}(l) - \mu_i(l)]^2. \tag{4.33}$$

The inter-class spread is the square of the distance between the centroids of the two classes,

$$\mathrm{d}^2(\boldsymbol{\mu}_1,\ \boldsymbol{\mu}_2) = \sum_{l=1}^{m} [\mu_1(l) - \mu_2(l)]^2. \tag{4.34}$$

A normalized measure of separateness between these two populations is the ratio of the average of the intra-class spread to the inter-class spread,

$$\frac{\sigma_1^2 + \sigma_2^2}{2\mathrm{d}^2(\boldsymbol{\mu}_1,\ \boldsymbol{\mu}_2)}. \tag{4.35}$$

The above quantities are based on multi-dimensional data vectors. If data sets over individual measurement variable are considered, $m = 1$ in

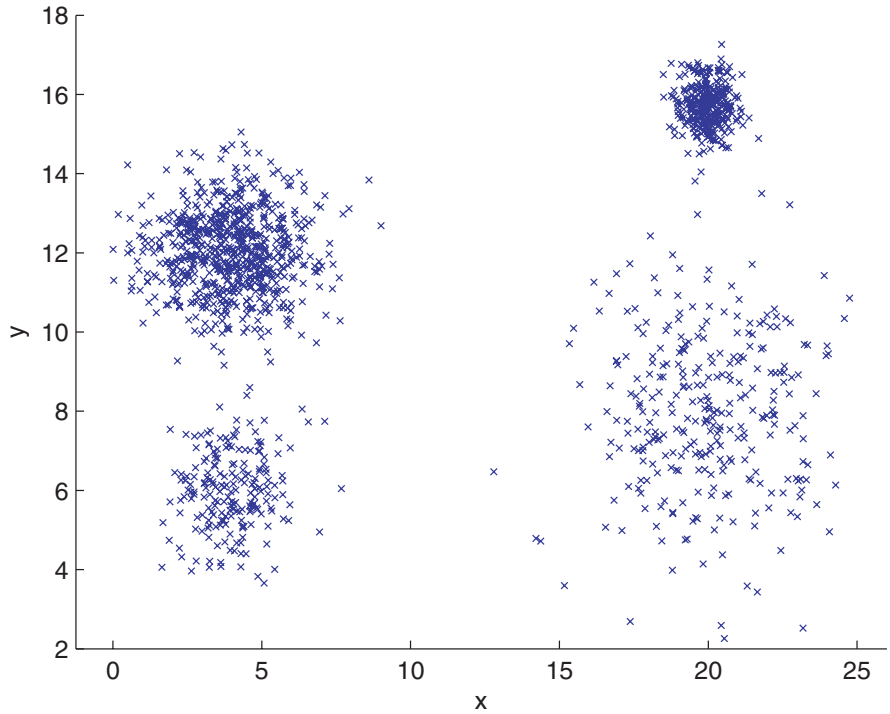


Fig. 4.18 A scatter plot of unlabeled samples well separated into two groups along x axis

equations (4.33–4.35). In the case of one dimension, the above normalized measure of separateness is invariant to scale. If the measure is less than one, the two groups of samples may be considered to be well separated. Figure 4.18 shows data samples that can be separated into two sets along the x axis by a threshold value of about 10.

4.9.2 Top-Down Method

Instead of combining individual samples into pairs, and larger groups up to a desired number of clusters, the overall data set can be considered to be a single group in the beginning and repeatedly divided until we have a desired number of clusters. This approach is known as divisive clustering. An advantage of divisive clustering over agglomerative clustering is that known information about inter-relationships can be incorporated into the divisive clustering algorithm. For example, in an application, approximate locations of the central portions of data vectors for one or more of the classes may be known. Alternatively, some examples of causes and effects may be known and these may be used to obtain a skeletal versions of trees for further refinement using many unlabeled samples. Kaufman and Rousseeuw (1990), present a divisive hierarchical clustering algorithm called the DIANA.

4.9.3 Interactive Approaches

In many practical applications, the decision tree design criterion is not well-specified at all. It is hoped that the above sections provide helpful approaches at various stages of data analysis and design of decision-making algorithms. Typically, the overall design may require some trial and error approaches and iterative refinements, at least at some stages of development. One example of these is eliminating outliers from consideration. One or a few data samples that lie significantly away from all other samples in the feature space can tilt the resulting solutions unreasonably. They may be eliminated from consideration. A second example of interactive changes is the pruning of decision trees. If a sub-tree covers very few samples, having a subtree for correct decisions within them may result in overfitting. Such a decision region may be merged with a

neighboring subtree. In the feature space, this amounts to combining small distinct regions with larger neighboring regions for simplification of the resulting tree.

4.10 Further Reading

The book by Duda et al. (2001), is a standard text and reference for pattern classification. It has detailed treatment of Bayesian decision theory, parameter estimation, linear discriminant functions, and clustering. It also has a section on support vector machines and two sections on decision trees. Anderberg (1973), and Kauffman and Rousseeuw (1990), are books on clustering. Breiman et al. (1984), is a book on developing trees from a purely statistical perspective. Decision trees are commonly covered in the wider topics of artificial intelligence (Russell and Norvig 2002) and data mining (Han and Kamber 2006). The book, by Quinlan (1993), is on programs for machine learning and has detailed discussions on many issues in decision tree development.

From time to time, survey articles on decision trees appear in literature. Most of these articles extensively cite and survey research reports on decision trees from numerous different points of view such as tree models from the nature of applications, logical approach to decision-making, problems in directory searching, computational complexity, their relations with neural networks, heuristics, and applications. Moret (1982), and Murthy (1998), are both extensive and scholarly surveys. The main emphasis in Moret (1982), is on representing Boolean functions as decision trees. On the other hand, Murthy (1998), is a more general survey of representation of data with decision trees. Other survey articles include the following. A synthesis of work on tree-structures in many application areas such as pattern recognition, decision tables, fault location, coding theory, and questionnaire design appears in Payne and Preece (1980). Dattatreya and Kanal (1985) is an overview of decision trees in pattern recognition. Various existing methods for designing decision tree classifiers and their potential advantages over single state classifiers are surveyed in Safavian and Landgrebe (1991).

The treatment of decision tree design in this chapter is introductory. Many specialized techniques for decision tree design have been developed and reported in the research literature. Some of these are application specific and some are general. These approaches are very helpful for an advanced study and for a serious designer in applications areas. Manago and Kodratoff (1991), develop an algorithm called KATE that learns decision trees from complex structured data. Evaluation of which feature to use at which point in the tree design is a well investigated problem. Rules for these are based on information theory, distance measures, or dependence measures (Ben-Bassat 1982). A more recent reference on feature selection measures is Kononenko and Hong (1997). Neural networks and decision trees are related approaches for decision-making. Such decision trees with parametric classifiers with small experts at decision-making nodes is studied by Jordan and Jacobs (1994). Chai et al. (1996), study genetic algorithms to develop linear splitting at decision-making nodes. Various methods for pruning trees have been suggested and investigated. Kim and Koehler (1994), analyze the conditions under which pruning improves the accuracy. Rastogi and Shim (1998), integrate decision tree construction with tree pruning.

Several commercial software packages are available for help with decision tree design, research, and applications. A good list of current software packages is available on the Knowledge Discovery and Nuggets web-page,

http://www.kdnuggets.com/software.html (4.36)

Within the class of commercial software for decision trees, there are two widely cited packages. One is CART, developed by Breiman et al. (1984), and upgraded from time to time. The second package is C5.0, developed following the book by Quinlan (1993). The older version of this, C4.5, is available as free software from the same web site. The web-site also has several other commercial packages as well as free software packages.

Following are recent articles that use decision tree principles for applications related to environmental sciences. Cannon and Whitfield (2002), use tree-based recursive partitioning models to develop mappings between synoptic-scale circulation fields and the leading linear and nonlinear principal components of weather elements observed at a surface station. Goel et al. (2003), use decision trees and artificial neural networks to identify weed stress and nitrogen status of corn. They use hyperspectral data from a

compact airborne spectrographic imager. Li and Claramunt (2006), design decision trees for the classification of geographical information. Their approach takes into account spatial autocorrelation phenomena in the classification process.

4.11 Conclusion

The practice of environmental sciences frequently encounters the problem of decision making in uncertainty. The task of decision making typically involves the measurement of some physical attributes and the assignment of a decision. The desired objective of decision making is to minimize some overall cost criterion. The mathematical relationships among the various aspects of the measurement and the decision making process govern the cost incurred through the process. The class of mathematical relationships includes the subclass of statistical relationships. In reality, the availability of such relationships are limited in three ways. All the aspects of relationships may not be available. Among those that are available, some may be represented ambiguously, instead of being represented accurately. Some of the relationships may be unambiguously represented but the representations may differ slightly from the corresponding real relationships. In addition to these drawbacks, development of good decision making algorithms may pose the following additional challenge. A conceived decision making algorithm may require prohibitive amount of computer time and/or memory. Artificial intelligence approaches help us to strike a compromise between (1) approximate representation of relationships, (2) computational complexities, and (3) the quality of the final solution to the problem at hand. Decision making through the use of decision trees is one such artificial intelligence approach. Quinlan (1990) focuses on techniques that discover classification rules based on knowledge about classified objects.

Decision trees are useful in many data analysis and classification applications, including medical diagnosis, meteorology, agriculture, pollution monitoring, and remote sensing. They are useful in applications in which hierarchical cause-effect relationships occur naturally. They are useful in applications with the peculiarity of a naturally recursive division of classes within classes. They are useful in applications in which expensive measurements are to be undertaken only if other readily available data so indicate. Finally, even in applications without any such model-driven reason for the use of decision trees, it may simply turn out to be convenient to design and implement decision-making algorithms as trees. Therefore, there are no global decision tree design approaches. A designer should use all available information about the nature of the problem, cause-effect relationships, and the structure of the training data for decision tree design. Ideas and approaches developed in this chapter can be used at any stage of the development of the decision-making algorithms, directly, or with modification. If decision regions are explicitly given, the design of a decision tree is conceptually simple. If possible decision boundaries are suggested, this information will help in the joint task of decision (pattern class label) assignment and decision tree design. If labeled or unlabeled training samples are given, the goal of designing a decision tree classifier influences the structure of decision boundaries. Approaches and techniques to jointly determine such decision boundaries and design decision trees are developed here.

The literature on Decision Trees is vast. Some recent books on pattern classification, clustering, artificial intelligence, and data mining are suggested for the interested reader. There are many survey articles on decision trees, some very general, and some with special emphasis. A few such articles are also cited. These are also invaluable resources of annotated bibliographies. Several research articles dealing with specific methods for decision tree design are included for readers interested in exploring advanced techniques. Recent research articles with applications of decision trees in areas related to environmental sciences are listed and suggested for the reader to gain an experimental perspective. Finally, many commercial software packages are available for the applied researcher. Location of these services on the world-wide web are given.

Acknowledgement The author thanks the reviewers for many constructive comments and Professor S. Lakshmivarahan for helpful suggestions throughout the preparation of this chapter.

References

Anderberg, M. R. (1973). *Cluster analysis for applications*. Academic: New York.

Ben-Bassat, M. (1982). Use of distance measures, information measures, and error bounds on feature evaluation. In P. R. Krishnaiah, & L. N. Kanal (Eds.), *Classification, pattern recognition, and reduction of Dimensionality. Handbook of statistics* (Vol. 2, pp. 773–791). Amsterdam: Elsevier Science.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth International Group.

Cannon, A. J., & Whitfield, P. H. (2002). 'Synoptic map-pattern classification using recursive partitioning and principal component analysis', *Monthly Weather Review*, *130*, 1187–1206.

Chai, B.-B., Zhuang, X., Zhao, Y., & Sklansky, J. (1996). Binary linear decision tree with genetic algorithm. *Proceedings of 13th International Conference on Pattern Recognition* (pp. 530–534). Los Alamitos, CA: IEEE Computer Society Press.

Dattatreya, G. R., & Kanal, L. N. (1985). Decision trees in pattern recognition. In L. N. Kanal, & A. Rosenfeld (Eds.), *Progress in pattern recognition* (pp. 189–237). Amsterdam: North-Holland.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification*. New York: Wiley-Interscience.

Goel, P. K., Prasher, S. O., Patel, R. M., Landry, J. A., Bonnell, R. B., & Viau, A. A. (2003). Classification of hyperspectral data by decision trees and artificial neural networks to identify weed stress and nitrogen status of corn. *Computers and Electronics in Agriculture*, *39*, 67–93.

Han, J., & Kamber M. (2006). *Data mining*. San Francisco: Morgan Kauffman.

Jordan, M. I., & Jacobs, R. A. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, *6*, 181–214.

Kauffman, L., & Rousseeuw, P. J. (1990). *Finding groups in data: An introduction to cluster analysis*. New York: Wiley.

Kim, B., & Koehler, G. J. (1994). An investigation on the conditions for pruning an induced binary tree. *European Journal of Operations Research*. *77*, 82–95.

Kononenko, I., & Hong, S. J. (1997). Attribute selection for modeling. *Future Generation Computer Systems*, *13*, 181–195.

Knowledge Discovery and Nuggets web-page, http://www.kdnuggets.com/software.html

Li, X., & Claramunt, A. (2006). A spatial entropy-based decision tree classification of geographical information. *Transactions in Geographical Information Systems*, *10*, 451–467.

Manago, M., & Kodratoff, Y. (1991). Induction of decision trees from complex structured data. In G. Piatetsky-Shapiro, & W. J. Frawley (Eds.), *Knowledge discovery in databases* (pp. 289–306). Cambridge, MA: AAAI/MIT Press.

Moret, B. M. E. (1982). Decision trees and diagrams. *ACM Computing Surveys*, *14*, 593–623.

Murthy, S. K. (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, *2*, 345–389.

Payne, H., & Meisel, W. (1977). An algorithm for constructing optimal binary decision trees. *IEEE Transactions on Computers*, *26*, 905–916.

Payne, R. W., & Preece, D. A. (1980). Identification keys and diagnostic tables. *Journal of Royal Statistical Society: Series A*, *143*, 253–292.

Quinlan, J. R. (1990). Decision trees and decision-making. *IEEE Transactions on Systems, Man, and Cybernetics*, *20*, 339–346.

Quinlan, J. R. (1993). *Programs for machine learning*. San Francisco: Morgan Kaufmann.

Rastogi, R., & Shim, K. (1998). Public: A decision tree classifier that integrates building and pruning. In A. Gupta, O. Shmueli, & J. Widom (Eds.), *Proceedings of the 24th International Conference on Very Large Data Bases* (pp. 404–415), San Francisco: Morgan Kaufmann.

Russell, S., & Norvig, P. (2002). *Artificial intelligence: A modern approach*. Englewood Cliffs, NJ: Prentice-Hall.

Safavian, S. R., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, *21*, 660–674.

Wilson, P. F, Dell, L. D., & Anderson, G. F. (1993). *Root cause analysis*. American Society for Quality: Milwaukee, WI.

Introduction to Genetic Algorithms

5

Sue Ellen Haupt

5.1 Motivation

The world is full of optimization problems. Nature constantly optimizes each of her configurations. Each ecosystem fits together to use the symbiotic nature of each element. Species have evolved to have the characteristics that are most likely to lead to survival. The wind blows in directions that best alleviate any imbalances in forces. The planets orbit in ways that best fulfill the laws of motion. In understanding the environment, we often have to discern the optimization problem to fully understand its solution.

Evolution is one of the most interesting optimization problems. Why have humans evolved to have two hands, two eyes, two legs, one head, and a large brain while other species have not? Does that make humanity the pinnacle of the optimization problem? Why do guppies evolve to have different characteristics in dissimilar environments? Can the process of evolution be codified to understand these issues better?

Many problems that we address in environmental science can be configured into an optimization problem. As an example, let's consider guppies evolving in an environment where they need to survive on the available food, attract mates for reproduction, and avoid predators. Figure 5.1 illustrates the pieces of a general optimization problem. We begin with input parameters that we wish to optimize. For our guppies, these variables might include attractiveness (to mates), disease tolerance, food requirements, appeal to their predators, and ability to hide from predators. The combination of the values of these variables makes the guppy an individual. There must be some method to rate the survivability of the guppy based on its specific variable values. There must be some way to balance its attractiveness to potential mates with its visibility to its predators. If the environment is harsh, it must be hardy. If the current is swift, it must have a long enough tail to swim fast. The objective, or cost, function codifies these considerations and weights them to rate the guppy survivability. The "most fit" guppies survive while those that do not meet the specifications of the objective function are destined to die off in a harsh environment or are eaten by a predator. The final piece of the optimization scheme is the optimization algorithm that finds some way of configuring new guppies so that they evolve into a viable species for their environment. The optimization algorithm typically minimizes some "cost," or equivalently, optimizes the objective.[1] Some common optimization algorithms include Newton's method for optimization, conjugate gradient, and Nelder-Mead downhill simplex method. Unfortunately, it is difficult to code guppy coloration, food requirements, and attractiveness to either mates or predators in a way to use these gradient seeking methods. More innovative methods are required.

A genetic algorithm (GA) is one such versatile optimization method. Figure 5.2 shows the optimization

Sue Ellen Haupt (✉)
Applied Research Laboratory and Meteorology Department, The Pennsylvania State University, P.O. Box 30, State College, PA 16802, USA
Phone: 814/863-7135; fax: 814/865-3287;
email: haupts2@asme.org

[1] In optimization terminology, an objective function could be either minimized or maximized. When the name cost function is applied, we always minimize. In contrast, a fitness function is maximized. Therefore, we concentrate on minimization problems. It is trivial to turn a maximization problem into one in minimization with a negative sign.

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

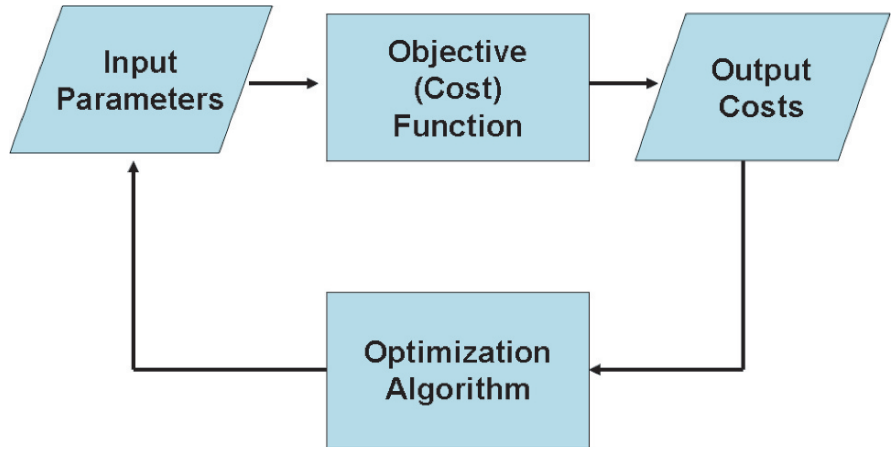


Fig. 5.1 Flowchart of the optimization process

process of a GA – the two primary operations are mating and mutation. The GA combines the best of the last generation through mating, in which parameter values are exchanged between parents to form offspring. Some of the parameters mutate. The objective function then judges the fitness of the new sets of parameters and the algorithm iterates until it converges. With these two operators, the GA is able to explore the full cost surface in order to avoid falling into local minima. At the same time, it exploits the best features of the last generation to converge to increasingly better parameter sets. GAs are remarkably robust and have been shown to solve difficult optimization problems that more traditional methods can not. Some of the advantages of GAs include:

- They are able to optimize disparate variables, whether they are inputs to analytic functions, experimental data, or numerical model output.
- They can optimize either real valued, binary variables, or integer variables.
- They can process a large number of variables.
- They can produce a list of best variables as well as the single best solution.

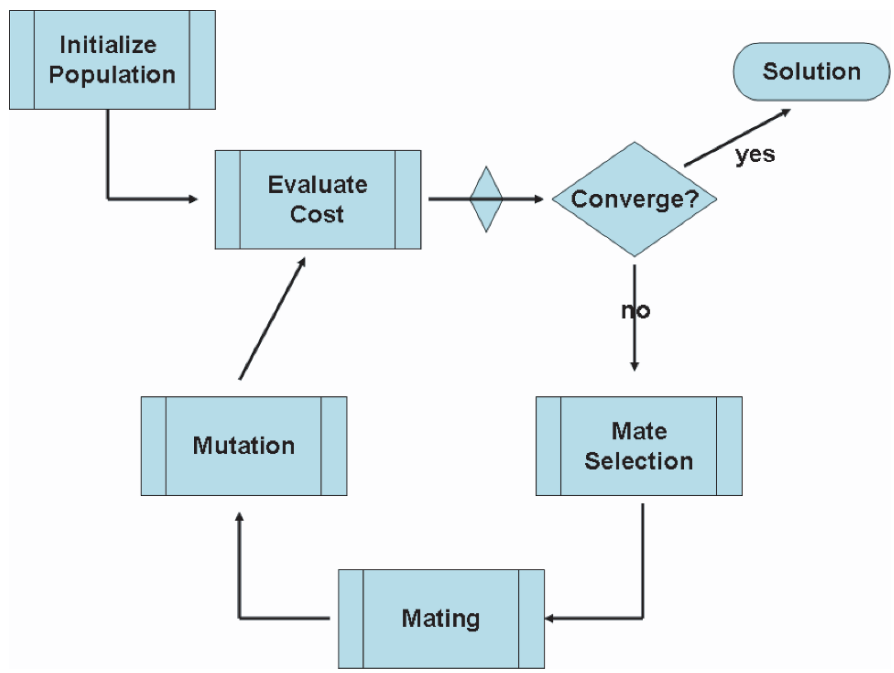


Fig. 5.2 Flowchart of optimization with a genetic algorithm

- They are good at finding a global minimum rather than local minima.
- They can simultaneously sample various portions of a cost surface.
- They are easily adapted to parallel computation.

Some disadvantages are the lack of viable convergence proofs and the fact that they are not known for their speed. As seen later in this chapter, speed can be gained by careful choice of GA parameters. Although mathematicians are concerned with convergence, often scientists and engineers are more interested in using a tool to find a better solution than obtained by other means. The GA is such a tool.

5.2 Genetic Algorithm Overview and Guppy Evolution

A genetic algorithm combines the concepts of genetics and evolution into an algorithm to optimize a function or to search a solution space. GAs were first introduced by John Holland (1975) at the University of Michigan (UM). David Goldberg (1989) popularized GAs beginning with his Ph.D. dissertation at UM where he used it to optimize a gasline piping problem, a problem that was quite difficult to solve via conventional means. De Jong (1975) demonstrated the utility of GAs for function optimization and studied how to best choose GA parameters. GAs have become a popular tool in the engineering literature but have thus far found fewer applications in the environmental sciences. More discussion of applications can be found in Chapters 14 and 18.

GAs can be configured as either binary or real valued. The GA literature began with the binary version, so we will start there too. Figure 5.2 depicts the optimization process for the GA. The basic process is the same as in Fig. 5.1, but now the GA operations of mating and mutation are specified.

The basic building block of a genetic algorithm is a gene, which represents each problem variable. For our guppy problem, the genes encode each characteristic of the guppy that we wish to consider. Table 5.1 shows the encoding of eight variables relevant to guppy evolution into eight genes. Each variable is encoded into a 2 bit gene, meaning that each variable can have up to four separate realizations. For instance, there may be four gradations of attractiveness to mates, in this case coded as:

11 = drop-dead gorgeous
10 = very handsome
01 = passable
00 = only if females are desperate

The genes are then concatenated to form a chromosome. The guppy encoded in Table 5.1 is then represented by a chromosome: 1001010001001100. The genetic algorithm is begun by creating a population of chromosomes using a random number generator. Since many computer languages generate random numbers between 0 and 1, for the binary GA the random number is simply rounded. We generate eight randomly configured guppies, each with eight 2 bit genes, and our population looks like:

$$pop = \begin{bmatrix} 1001010001001100 \\ 0110001001001110 \\ 0000111010101110 \\ 0101001110010111 \\ 0011000111001010 \\ 1100011010001101 \\ 0001100110100111 \\ 0110011001110010 \end{bmatrix} \tag{5.1}$$

In this matrix, each row represents a chromosome, or complete guppy configuration.

The fitness of each member of the population is then evaluated via the cost function. For the guppy example, the cost function depends on the environment.

Table 5.1 Example guppy encoded into binary genes

Attractiveness (to other guppies)	Tail	Attractiveness (to predator)	Dappling (blending into environment)	Temperature tolerance	Disease tolerance	Food requirements (amount)	Feeding requirements (time between)
Very handsome	Short	Tasty	None	Little	Hardy	Bottomless pit	Frequent feeder
10	01	01	00	01	00	11	00

Each gene of the guppy is first assigned an adaptation value. For instance, the attractiveness gene, which determines the guppy's likelihood to attract a mate, is assigned adaptation values denoted in the MATLAB code below:

```
attractive = x(:,1:2); %grabs the first two bits which
   form the first gene
%likeliness to mate
%attractiveness (brightness positive)
if attractive(ind,:)==[1 1]
adapt(ind,1)=2.0; %drop dead gorgeous
elseif attractive(ind,:)==[1 0]
adapt(ind,1)=1.5; %very handsome
elseif attractive(ind,:)==[0 1]
adapt(ind,1)=1.0; %passable
else %[0 0]
adapt(ind,1)=0.5; %if the female guppies are
   desperate
end
```

In this case, the more attractive the guppy, the higher adaptation value is assigned. The second aspect of the guppy cost function weights the importance of each gene for survivability in a particular environment. Each adaptation value is weighted according to how likely that characteristic will result in the guppy (1) mating or (2) being eaten by a predator. Specifically, for a bright, well lighted pool with lots of predators, weights are assigned as:

```
%habitat 1
wts(1,:)=[1.0, 0.8, 0.0, 0.7, 0.0, 0.0, 0.0, 0.0];
   %probability of mating
wts(2,:)=[0.0, 0.0, 0.5, 0.8, 0.6, 0.2, 0.9, 0.4]
   %probability of getting eaten
```

The final step of judging the adaptability of each guppy is writing a cost function that multiplies the adaptability values for the guppy by the environment weights and weighting the importance of mating versus getting eaten for this habitat:

```
f = − ( (wts(1,:)*adapt')' + 3*(wts(2,:)*adapt')' );
```

Note that a negative sign is applied to the cost function since the GA routine is configured to look for minima. Each member of the guppy population is judged via the cost function and the costs assigned as:

$$\text{Cost}\begin{bmatrix}1001010001001100\\0110001001001110\\0000111010101110\\0101001110010111\\0011000111001010\\1100011010001101\\0001100110100111\\0110011001110010\end{bmatrix}=\begin{bmatrix}-5.4\\-4.7\\-2.1\\-4.2\\-6.7\\-7.3\\-5.9\\-1.8\end{bmatrix} \tag{5.2}$$

The next step is simply sorting the costs with the smallest cost (most fit) chromosomes put at the top:

$$\text{Cost}\begin{bmatrix}1100011010001101\\0011000111001010\\0001100110100111\\1001010001001100\\0110001001001110\\0101001110010111\\0000111010101110\\0110011001110010\end{bmatrix}=\begin{bmatrix}-7.3\\-6.7\\-5.9\\-5.4\\-4.7\\-4.2\\-2.1\\-1.8\end{bmatrix} \tag{5.3}$$

Now we are ready for the natural selection to occur. In terms of the guppies, the less fit half of the population is eaten by predators or simply dies without reproducing. We are left with only the top half (most fit four) of the population matrix above.

The GA operations of mating and mutation come into play. In mating, we select two members of the population to exchange information to produce offspring. For the guppy problem, we will use tournament selection. Here, three members of the population are randomly selected and the two most fit individuals (smallest cost function values) of that tournament will then mate. The algorithmic mating procedure mimics the genetic recombination of meiosis. In meiosis, the chromosomes line up and join at a kinetochore. When the chromosomes separate, the left portion of the mother chromosome conjoins with the right portion of the father chromosome to complete the process known as crossover. In our binary chromosomes, the process is equivalent. A random kinetochore, or crossover point, is selected and the genes to the left of this point on parent 1 are concatenated with those to the right of that point on parent 2. In this case, we form two new individuals. An example of the guppy chromosomes mating is:

$$\textbf{cost}\begin{bmatrix}\mathbf{11000110}10001101\\ \mathbf{00110001}1001010\end{bmatrix}=\begin{bmatrix}-7.3\\ -6.7\end{bmatrix}$$

$$\textbf{cost}\begin{bmatrix}\mathbf{11000110}1001010\\ \mathbf{00110001}10001101\end{bmatrix}=\begin{bmatrix}-7.4\\ -3.7\end{bmatrix}$$

We see that one of the offspring guppies is more fit than either parent and one is less fit.

The second GA operation is mutation. Before mutating, we typically apply elitism and set aside the most fit (lowest cost) individual of the entire population and do not allow it to mutate. Then we go into the matrix and randomly change the bit value of a predetermined percentage of bits. After mating and mutation, the guppy population looks like:

$$\text{cost}\begin{bmatrix}\textit{1100011010001101}\\ 0011000111001010\\ 0001100110100111\\ 1001010001001100\\ 1100011011001010\\ 0011000110001101\\ 0001100110100111\\ 0011000111001010\end{bmatrix}=\begin{bmatrix}\textit{−7.3}\\ -6.7\\ -7.6\\ -5.4\\ -5.2\\ -3.7\\ -7.7\\ -2.1\end{bmatrix}\qquad(5.4)$$

The first row (italics) is the best "elite" guppy chromosome. The next three are the other parents that remain in the population from the last generation, but have now been subject to bit changes due to mutation (bold). The last four rows are the offspring guppy chromosomes after mutation. The first iteration has completed. The process is iterated until convergence is obtained. For the guppies, convergence means a stable population with all individuals about equally adapted (i.e. the average population is fairly stable). The lowest cost individual becomes a prototype for the guppy population with some variation around it. Figure 5.3 shows the convergence for a run of the guppy problem using a population size of 16, crossover rate of 0.5, and mutation rate of 0.2. The "best" guppy is identified after about 11 generations, but it takes a bit longer for the population to stabilize. Convergence is both problem dependent and run dependent. Since the GA relies on random numbers to generate problems and perform the mating and mutation operations, each run of the GA will produce slightly different results. Often, the primary difference is how many iterations are required to produce convergence.

This example merely serves as a basic introduction to GAs. Many variations are possible, often

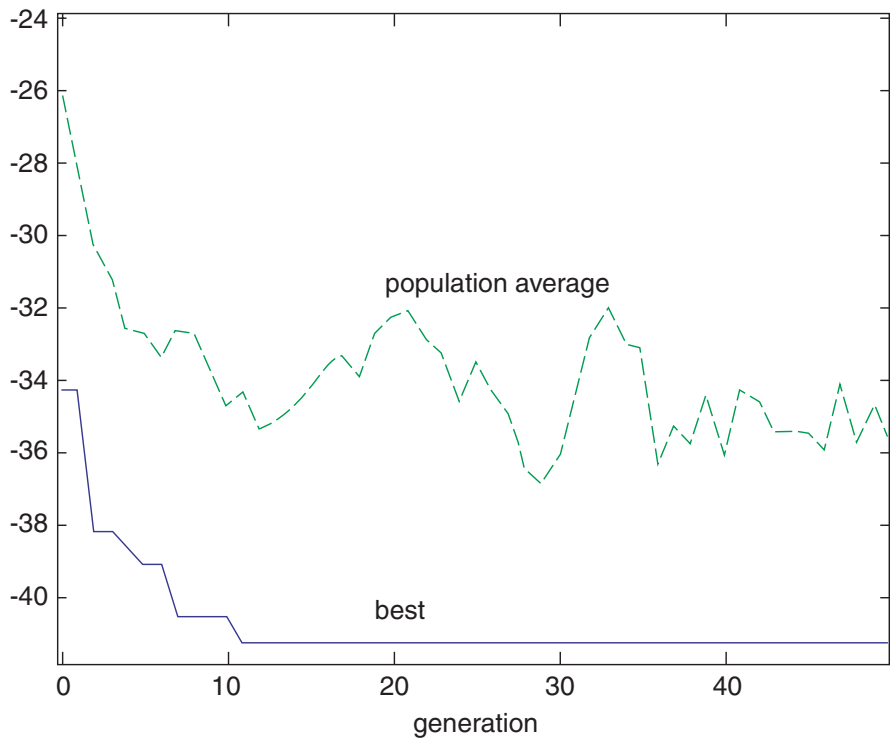


Fig. 5.3 Convergence of the GA for the guppy evolution problem

interpretable as limiting branches of the basic GA discussed here. For instance, some GAs are purely "asexual," that is, they reproduce without a mating operator, using only mutation. That scenario is equivalent to using a crossover rate of zero. Tuning the GA parameters will be discussed in Section 5.5.

5.3 Binary Genetic Algorithms and Function Optimization

We saw how GAs can be applied to a problem in evolution where the use of a GA is more "obvious." More often, we have some function we wish to optimize, subject to constraints. Here, we'll examine solving a specific function of two variables and use this example to observe the process of convergence in more detail. We wish to minimize:

$$f(x, y) = \sin(x) J_1(y) \tag{5.5}$$

where J_1 is the Bessel function of the first kind of order 1. We'll call this function, "Besin." Equation (5.5) is solved subject to the constraints:

$$0 \leq x \leq 10$$
$$0 \leq y \leq 10$$

Figure 5.4 shows plots of the equation on the given interval using both a three dimensional view and a contour plot. The exact solution, $(x, y) = (4.71, 1.81)$, is indicated on the plots.

The first step of solving our "Besin" optimization problem with a GA is initializing the population. We choose to code our binary GA with an 8 bit representation of each of the two (x, y) variables. For instance, the 16 bit chromosome with two genes representing x and y is:

$$[0\,1\,1\,1\,0\,0\,0\,0\,0\,1\,0\,1\,0\,0\,0\,0] = (4.3922, 3.1373)$$

The initial population of 16 chromosomes is shown in Fig. 5.5 as asterisks on the contour plot. Costs of each (x,y) point are computed via (5.5). The minimum cost of this initial population is -0.46082 and the mean cost is 0.0096951.

The solution is evolved by the GA using a crossover rate of 0.5, population size of 16, and mutation rate of 0.2. Figure 5.6 shows the next four iterations of the GA, the sixth iteration, and the fortieth iteration. We see that the GA initially explores the entire solution space, particularly in the local minima. By the sixth iteration, the best chromosome is near the exact solution and by the fortieth iteration many of the population members are in the global solution well.

Table 5.2 Convergence of the GA solution of (5.5)

Iteration	Min cost	Mean cost	x	y
Initial	−0.46082	0.0096951	5.3333	1.5686
1	−0.46082	−0.064677	5.3333	1.5686
2	−0.46082	−0.084943	5.3333	1.5686
3	−0.50994	−0.12589	4.6275	1.2549
4	−0.50994	−0.044344	4.6275	1.2549
5	−0.57378	−0.023693	4.5490	1.8824
10	−0.58001	−0.23114	4.7843	1.8824
15	−0.58151	−0.11361	4.7059	1.8824
Exact	**−0.58186**		**4.71**	**1.81**

Table 5.2 shows the convergence for this problem. We see a rather good convergence after only 5 iterations and total convergence after 15 iterations. Notice that although the best solution converges rather quickly, the GA continues to explore the solution space. Therefore, the mean cost converges rather slowly. In fact, Fig. 5.7 plots the convergence of the solution. We see that the population average cost continues to oscillate. This behavior is due to the large mutation rate specifically chosen to force continued exploration of the solution space for this wildly oscillatory function. Thus we conclude that the choice of GA population and mutation parameters affects the performance of the algorithm. This is most certainly true and will be discussed in more detail below. But first, let's look at another way to solve this problem – directly using the real values of the (x, y) coordinates.

5.4 Continuous Variable GA – Application to Optimization

We began with the binary GA because that is where the field started. This beginning also helps explain why certain methods are used for the operations of mating and mutation that we do when working with real numbers. Plus there are problems, like the guppy evolution example, where choices are not in terms of real-valued variables. Many of the problems that we encounter, however, involve optimizing real number continuous variables, so why not work directly with continuous variables and dispense with coding in binary? In fact,

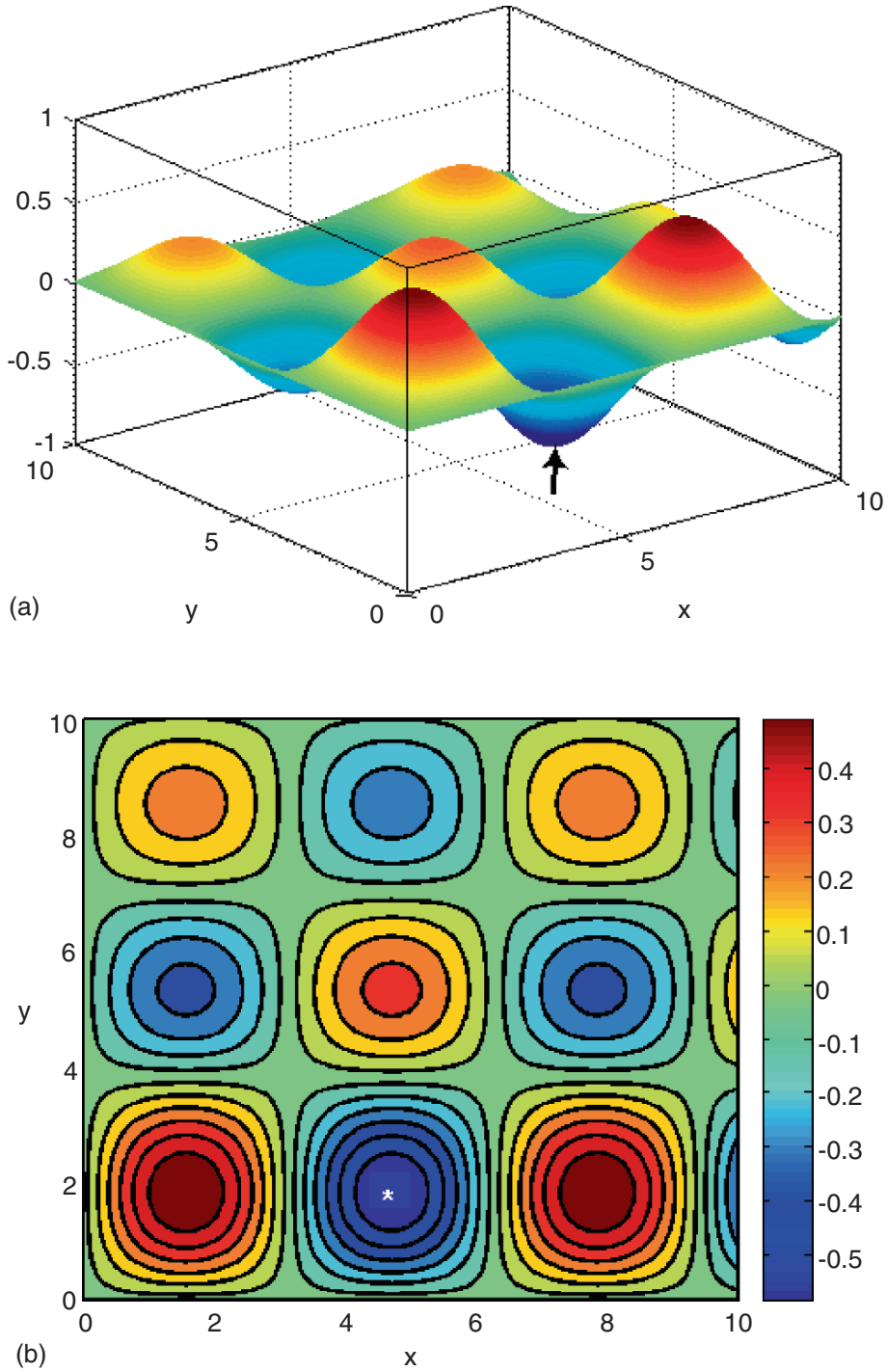


Fig. 5.4 Three dimensional (a) and contour (b) plots of the function of (5.5)

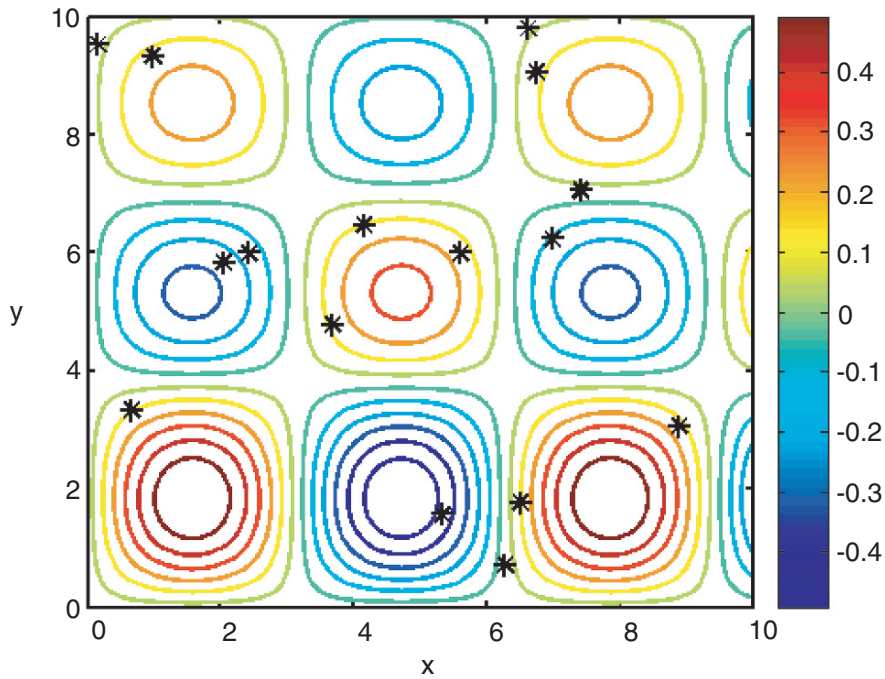


Fig. 5.5 Initial population for the binary GA solution to (5.5) superimposed on a contour plot of the function value

that is precisely what is now done with many real-valued problems.

The continuous variable GA works much the same as its binary cousin. Figure 5.8 is a flowchart of the continuous GA. It looks much the same as the flowchart for the binary GA (Fig. 5.2). The only differences are in the way certain operations are performed. In particular, chromosome definition, application of the cost function, and implementation of the operations of mating and mutation are modified to be appropriate for real values and are the only substantative changes. Let's look at each of these in more depth.

When initializing the population for continuous optimization with a GA, we must initialize the chromosome for the number of variables that we wish to find. In particular, given N_{par} variables to initialize, we define a chromosome consisting of parameters or variables each denoted by p_i as:

$$\text{chromosome} = \left[p_1, p_2, p_3, \cdots p_{N_{\text{par}}}\right] \tag{5.6}$$

Thus, when we define the cost function, it is in terms of a function of those variables:

$$\text{cost} = \text{F(chromosome)} = F\left[p_1, p_2, p_3, \cdots p_{N_{\text{par}}}\right] \tag{5.7}$$

The operations of mating and mutation are also altered to take into account these real-valued continuous variables. There are numerous methods of mating and mutation, but the ones demonstrated here are rather straightforward and most closely mimic the binary versions. For mating, the first step is to randomly choose the crossover point. The parents are then selected according to some selection criterion and labeled as m and d (mom and dad):

$$\begin{aligned} parent_m &= \left[p_{m1}\, p_{m2} \cdots p_{m\alpha} \cdots p_{mN_{\text{par}}}\right] \\ parent_d &= \left[p_{d1}\, p_{d2} \cdots p_{d\alpha} \cdots p_{dN_{\text{par}}}\right] \end{aligned} \tag{5.8}$$

The process of crossover then blends the information from the two parents. A way that most closely matches the binary GA swaps the portions of the chromosome to the right of the crossover point and blends the variable chosen as the crossover point (kinetochore):

$$\begin{aligned} p_{new1} &= p_{m\alpha} - \beta\left[p_{m\alpha} - p_{d\alpha}\right] \\ p_{new2} &= p_{d\alpha} - \beta\left[p_{m\alpha} - p_{d\alpha}\right] \end{aligned} \tag{5.9}$$

Here, β is the blending parameter between 0 and 1. The result is offspring of the form:

$$\begin{aligned} offspring_1 &= \left[p_{m1}\, p_{m2} \cdots p_{\text{new1}} \cdots p_{dN_{\text{par}}}\right] \\ offspring_2 &= \left[p_{d1}\, p_{d2} \cdots p_{\text{new2}} \cdots p_{mN_{\text{par}}}\right] \end{aligned} \tag{5.10}$$

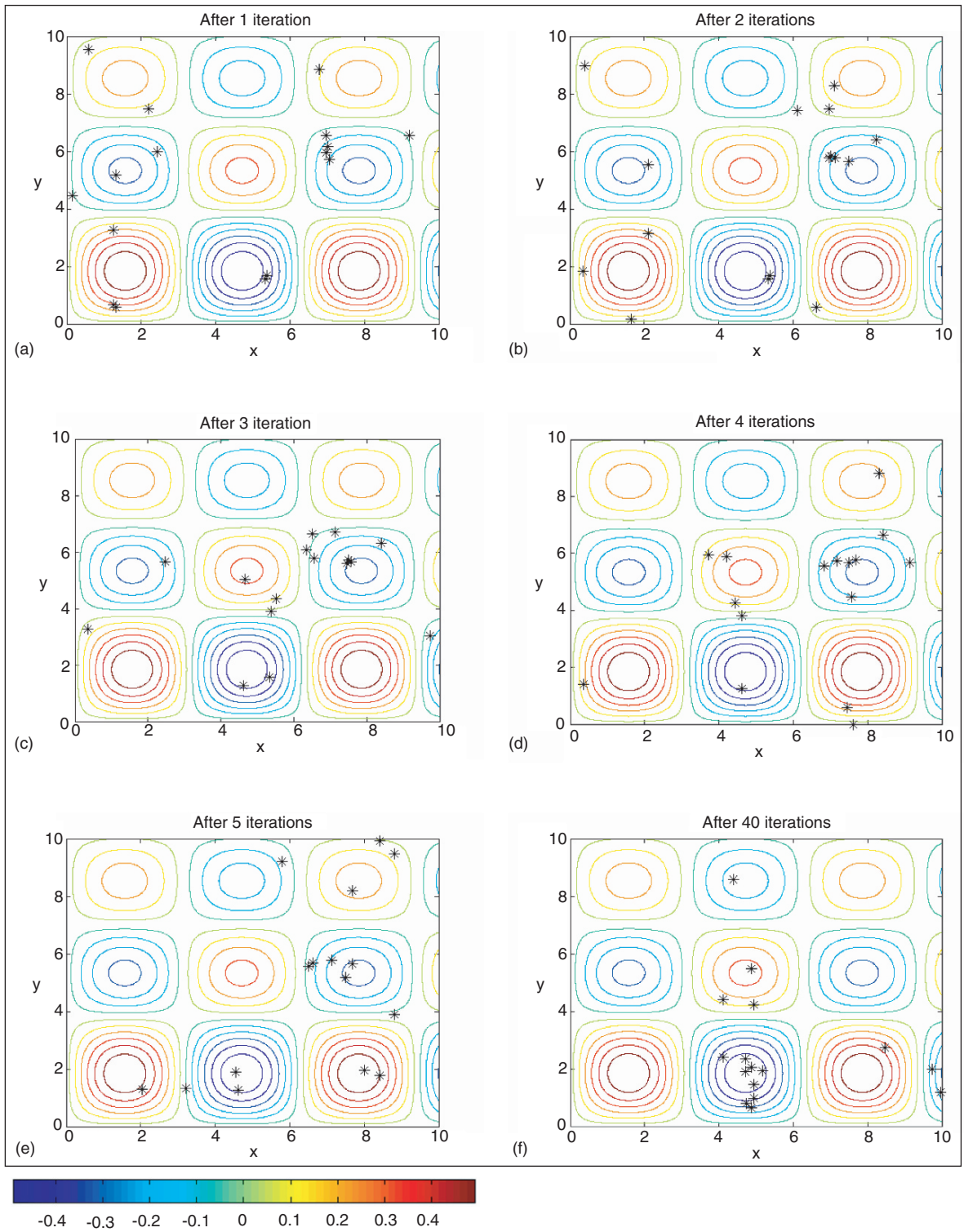


Fig. 5.6 Evolution of the GA population for solution of (5.5)

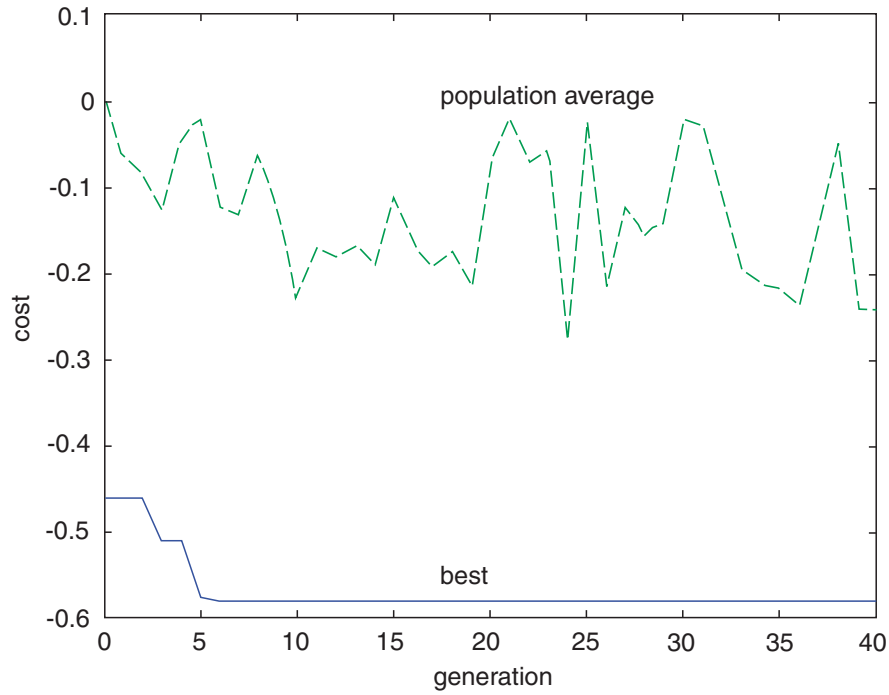


Fig. 5.7 Plot of convergence of the solution of (5.5). The solid line is the single best solution and the dotted line depicts the population mean

Altering the mutation operator is even more simple. For a continuous GA, we merely generate a new random number, $p_{i_{\text{new}}}$, to replace the original value. So if the original chromosome is

$$\textit{chromosome} = \left[p_1, p_2, p_3, p_4, \cdots p_{N\text{var}}\right] \tag{5.11}$$

and we wish to mutate the third parameter, the new chromosome will look like

$$\textit{mutated chromosome} = \left[p_1, p_2, p_{3_{\text{new}}}, p_4, \cdots p_{N\text{var}}\right] \tag{5.12}$$

Let's revisit the solution of (5.5) with the continuous GA. We choose to use a population size of 12,

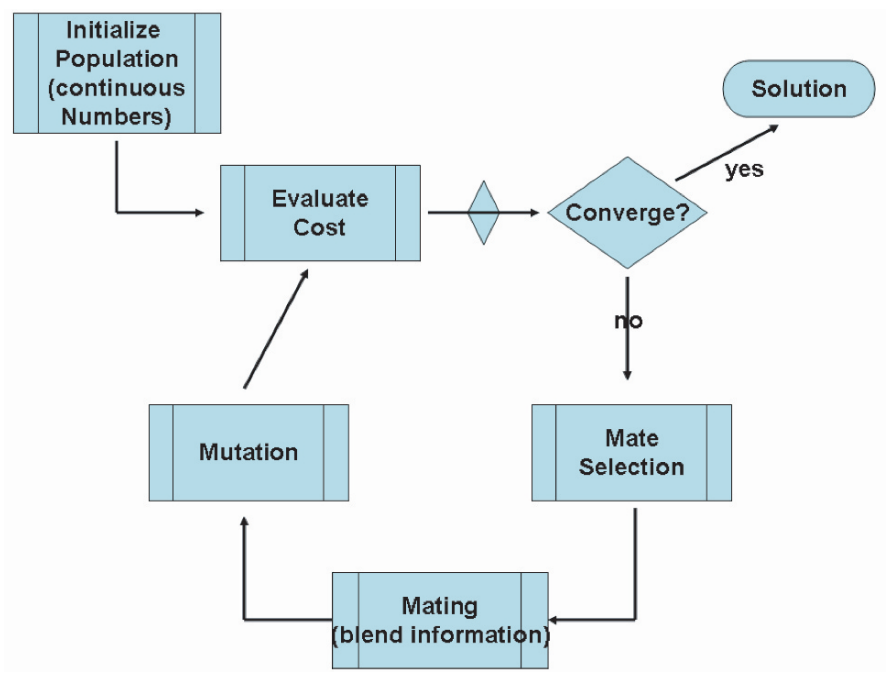


Fig. 5.8 Flow chart of a continuous parameter GA

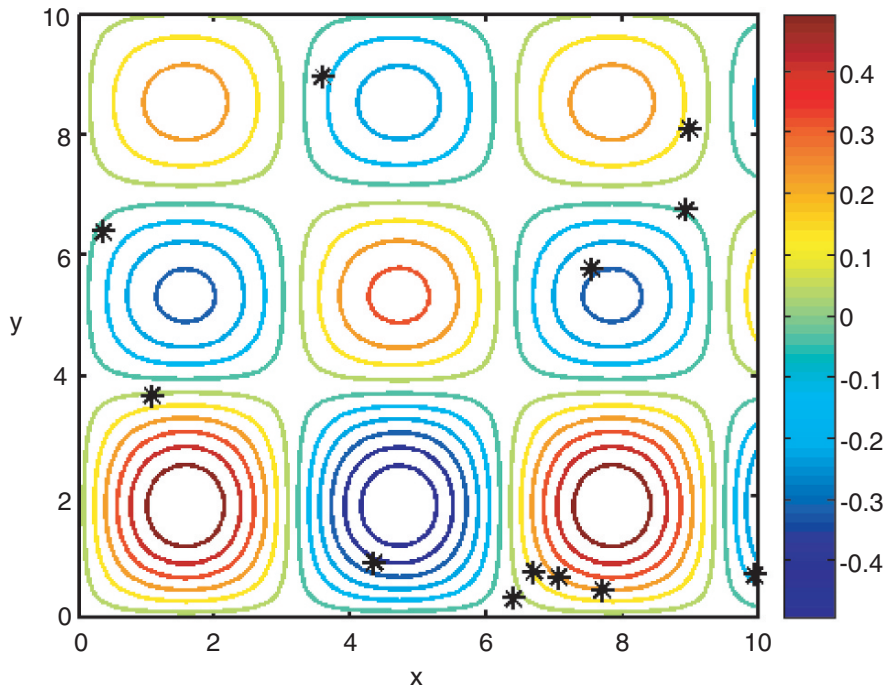


Fig. 5.9 Initial population of solutions to (5.5) for the continuous GA problem superimposed on a contour plot of the function

Table 5.3 Initial population (guesses) of solutions to (5.5) for the continuous GA and their associated costs

x	y	Cost
7.7109	0.4337	0.20967
6.7016	0.7454	0.14116
3.5830	8.9600	−0.10674
9.9596	0.6927	−0.16617
1.0709	3.6673	0.05914
8.9394	6.7496	−0.03752
4.3533	0.8984	−0.37954
6.3978	0.3043	0.01720
7.5431	5.7726	−0.29981
7.0620	0.6602	0.21947
0.3515	6.3859	−0.06386
8.9856	8.0849	0.10452

mutation rate of 0.2 and crossover rate of 0.5. Table 5.3 indicates the initial population which is also plotted on the contour plot of Fig. 5.9.

Let's follow the mating process carefully for this example. The population of Table 5.3 is sorted and the least fit chromosomes are discarded. This process leaves the following chromosomes:

4.3533	0.8985
7.5431	5.7726
9.9596	0.6928
3.5830	8.9600
0.3516	6.3859
8.9394	6.7496

These remaining chromosomes are then ranked for mating. The probability of mating is determined here by rank weighting. To do that we make a table of probabilities of mating and the cumulative probability as:

probability of mating	cum probability
0.28571	0.28571
0.2381	0.52381
0.19048	0.71429
0.14286	0.85714
0.095238	0.95238
0.0476191	1.00000

We now use a roulette wheel selection process where a random number generator determines

$$\text{pick1} = 0.26006$$
$$\text{pick2} = 0.89538$$

These correspond to chromosomes ranked 1 and 5. Now the random number generator is applied again to choose the crossover point, which is chosen here in the second variable. The value of β is randomly chosen as 0.32178. So the new offspring generated and their associated costs are:

	x	y	cost
Offspring 1:	4.3533	2.6642	−0.2689
Offspring 2:	0.3516	4.6202	−0.2496

The next step is mutation, that is, randomly changing some of the values of the parameters. Table 5.4 shows the new population. Highlighted values denote

Table 5.4 The new population at iteration 1 of solving (5.5) with a continuous parameter GA. Mating and mutation have altered the population. Highlighted values denote mutations. The original six chromosomes do not mutate, but 4 out of the 12 values in the six offspring mutate this generation

Original x	Original y	Original cost	Mutated x	Mutated y	Mutated cost
4.3533	0.8985	−0.37954	4.3533	0.8985	−0.37954
7.5431	5.7726	−0.29981	7.5431	5.7726	−0.29981
9.9596	0.6928	−0.16617	9.9596	0.6928	−0.16893
3.5830	8.9600	−0.10674	3.5830	8.9600	−0.10674
0.3516	6.3859	−0.06387	0.3516	6.3859	−0.06387
8.9394	6.7496	−0.03753	8.9394	6.7496	−0.03753
4.3533	**2.6642**	**−0.42353**	**7.2802**	**4.9394**	**−0.26890**
0.3516	4.6202	**−0.09000**	**0.8155**	4.6202	−0.24959
0.3516	5.7588	−0.10909	0.3516	5.7588	−0.10909
9.9596	1.3199	**−0.26825**	**0.3702**	1.3199	**0.19042**
4.3533	3.5033	−0.12733	4.3533	3.5033	−0.12733
3.5830	6.3552	0.08275	0.3583	6.3552	0.08275

mutations. The original six chromosomes are not mutated here, but four of the twelve values mutate in the new generation.

The continuous GA was iterated to complete the solution process. The correct solution was found after 10 iterations. Figure 5.10 shows the population clustered around the exact solution at the completion of the tenth iteration. Figure 5.11 denotes convergence. We see that the cost function value for this particular run tended to generally decrease for the entire population rather than just for the best individual.

We must be careful, however, about overinterpreting convergence. We should always remember that, because the GA uses random numbers for its initialization and operations, every time we run the GA we'll obtain a somewhat different result. If we have carefully chosen our parameters and done enough generations, we usually converge to the correct solution. Occasionally, however, our luck will be bad and we may either not get to the solution in the allowed number of iterations or convergence will be slow. Each set of initial parameters, each mating operation,

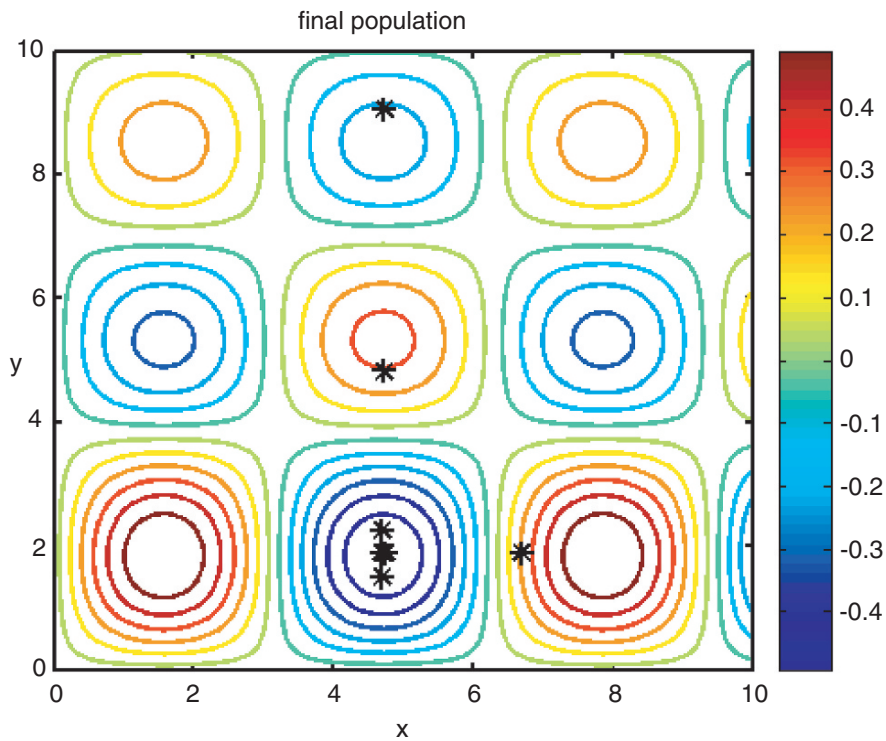


Fig. 5.10 Final population after 10 generations cluster near the exact solution to (5.5)

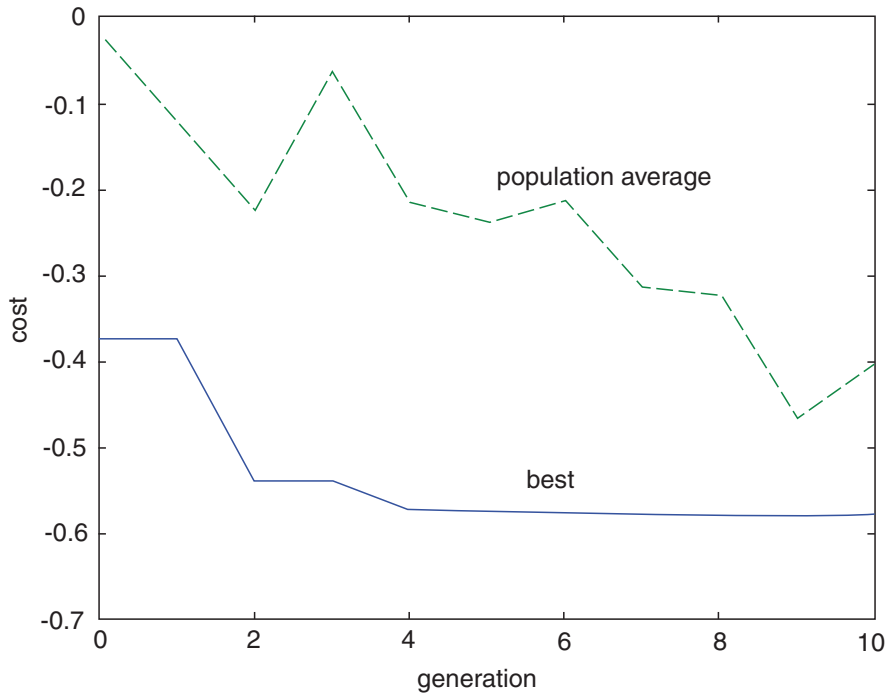


Fig. 5.11 Convergence of the continuous GA solution to the Besin problem (5.5)

and each application of mutation will have different results: therefore, the convergence plot and behavior of the solutions will differ with each run. So how do we assure ourselves that the GA is performing adequately? The answer is that we recognize these issues and deal with them directly. We never fully believe the results of a single GA run. Instead, we make a habit of always doing repeated applications. When we run the GA 10 times with essentially the same result, we are assured that the GA is behaving solidly for this problem and the solution is well optimized.

5.5 Optimizing GA Application

We have now observed several applications of a GA. So far, we have arbitrarily chosen performance parameters such crossover rate, crossover methodology, mutation rate, and population size. Let's look at these GA tuning parameters in a bit more detail.

5.5.1 Continuous vs. Binary GA

First, we have used both binary and continuous variable GAs. Which one is better? The answer, of course, depends on the problem. When we have either/or choices or choices between a small number of potential realizations (Is the stream of the guppies' environment rocky or smooth? Is the guppy dappled or plain?) then the binary GA is most appropriate. If we instead are searching for all real numbers, then one might as well use a continuous GA. Of course, one can always code continuous numbers in binary notation, but in general, the resulting GA is not as efficient as the continuous form. What if our parameters are mixed – we have some that are more conducive to binary coding and others toward continuous? We can always use a binary GA with the continuous variables coded into binary. New methods that mix the binary with the continuous have recently been developed and show promise for very efficient GA application (Haupt 2007, Haupt et al. 2008).

5.5.2 Variable Normalization

A related issue is whether and how to normalize our variables. If all of our variables are similar (for example, if we're on an (x, y) Cartesian grid from 5 to 10), normalization is irrelevant. The behavior of the GA does not depend on whether every number in the problem is divided by the same value. If however,

we have variables that are wildly disparate in nature, we may want to normalize them to be on the same order of magnitude. If we're on an (x, y) Cartesian grid where x varies from 10^{-6} to 10^{-5} and y varies from 10^5 to 10^6, then we would want to divide all x's by the mean x and all y's by the mean y before the search and cost function evaluation. One common technique is to always normalize all variables to a scale from 0 to 1. This approach has several advantages: (1) it assures that the behavior of the GA is not biased to any one variable, (2) it avoids normalizing the variable each time it is sent to a cost function, and (3) it makes generating random numbers rather easy since most random number generators give numbers between 0 and 1. For some problems, it may make more sense to map a variable with some function before doing a GA application. An example from air pollution in Chapter 14 maps the monitored concentrations onto a logarithmic scale for GA solution.

5.5.3 Mate Selection

The methods of selecting mates vary rather widely. The simplest methods are random without respect to any weighting. For instance, one can just randomly choose two parents to mate. This method is known as unweighted roulette wheel pairing. The more prevalent methods are based on either a rank or cost weighting method. The most basic ranked method is to pair the sorted chromosomes in order – the first chromosome mates with the second, the third with the fourth, and so on. The more refined ranked methods compute the probability of mating according to the ordering after the chromosomes are sorted from best to worst. That was the method demonstrated in Section 5.4. Probability of mating varies according to cumulative order of the rank. Cost weighting is computed according to the actual values of the cost function – a very low cost chromosome is much more likely to be selected for mating than the one that may be next in line. The costs are merely summed, then each cost divided by the total to obtain its probability of selection.

Once the selection probabilities are computed via either ranked weighting or cost weighting, the selection can occur via any of several methods. A common method is the roulette wheel selection (demonstrated in Section 5.4) where a random number is chosen and mapped to the cumulative probability of the ranked or cost weighted chromosomes. Another common method is tournament selection (demonstrated in Section 5.2). Three (or some other number) potential parents are selected according to random number selection matched to the computed probability of mating. The two with the lowest costs then mate. One can go into much more detail on the mate selection methods and we refer the reader to any of the GA books to read more (Goldberg 1989; Mitchell 1996; Davis 1991; Haupt and Haupt 2004).

5.5.4 Mating

Once the mates are selected, the crossover technique must be specified. The first decision regards the crossover rate, that is, how many population members should be replaced with offspring at each generation. In general, although some investigators have looked at this sensitivity, it doesn't make much impact on GA performance what rate is used. Using a crossover rate (X_{rate}) of 0.5 so that the number kept (N_{keep}) is half of the population size ($N_{\text{keep}} = X_{\text{rate}} \times N_{\text{pop}}$) is as good as any and is the typical choice.

The next issue is how to perform the crossover. The examples above used single point crossover. For the binary GA, it is easy to extend the crossover to two points, three points, or use three parents in a rather straightforward way. Uniform crossover can be performed by creating a random mask of 0s and 1s to determine whether to use the value in the mother or the father chromosome at each element.

For the continuous parameter GA, equations (5.9) and (5.10) blend the information along the axes of the parent chromosomes. When the blending parameter, β, is less than or equal to 1, the values of the offspring will be between the parents. When β is allowed to be greater than 1, then the axes are extended beyond the magnitude of the parent values, which is sometimes a good choice. A more simplistic continuous GA mating scheme merely swaps values of the real-encoded genes between the parents. If the parents are denoted as in (5.8), and we choose to swap between genes 2 and 5, we would obtain

$$\begin{aligned} offspring_1 = & \\ & [p_{m1},\ p_{m2},\ \uparrow p_{d3},\ p_{d4},\ \uparrow p_{m5},\ p_{m6}, \ldots, p_{mN_{\text{var}}}] \\ offspring_2 = & \\ & [p_{d1},\ p_{d2},\ \uparrow\ p_{m3},\ p_{m4},\ \uparrow p_{d5},\ p_{d6}, \ldots, p_{dN_{\text{var}}}] \end{aligned} \tag{5.13}$$

The problem with this simple scheme is that no new values are ever generated. The opposite extreme is uniform random crossover where each gene is assigned a random blending parameter β_i and the resulting offspring are constructed as

$$\begin{aligned} offspring_1 &= parent_1 - [\beta_1(p_{m1} - p_{d1}), \\ &\quad \beta_2(p_{m2} - p_{d2}), \cdots, \beta_{N_{\text{var}}}(p_{mN_{\text{var}}} - p_{dN_{\text{var}}})] \\ offspring_2 &= parent_2 + [\beta_1(p_{m1} - p_{d1}), \\ &\quad \beta_2(p_{m2} - p_{d2}), \cdots, \beta_{N_{\text{var}}}(p_{mN_{\text{var}}} - p_{dN_{\text{var}}})] \end{aligned} \tag{5.14}$$

A scheme similar to this was used for extended runs of the air pollution source characterization problem of Chapter 14.

5.5.5 Choosing Population Size and Mutation Rate

The choice of the GA parameters of population size and mutation rate can make a large impact on algorithm performance. The performance measure that we will concentrate on here is the number of cost function evaluations required to meet a pre-specified tolerance level of accuracy of the solution. We prefer this measure because: (1) it is easy to keep track of how many times the cost function has been called, (2) as applied scientists, we often want to find the "best solution": thus, the number of calls to find that best solution is the relevant quantity in measuring required computer time, and (3) it is not dependent on the type of computer being used.[2] Of course, measuring function calls does not represent all aspects of the GA (such as population generation, mating, mutation, sorting, etc.). For large problems that are computationally intensive, however, the number of function evaluations is often the controlling factor for measuring GA speed. Since the GA begins with random numbers, each new run of the GA will take a different number of function evaluations to "solve" the problem.

[2] Note that we are assuming a serial computer for this discussion. Application on parallel machines is beyond the scope of this chapter. Application on a parallel machine will change the performance as a function of mutation rate and population size, depending on how the GA is adapted to optimize the parallelism of the type of machine being used.

We present a sensitivity analysis of the number of function evaluations necessary to solve the Besin problem (5.5) for both the binary and continuous GA. Since we know the exact solution for this construed problem, we can easily determine convergence. Here, we stop the GA when the error in the solution becomes less than 5×10^{-3} or we surpass 5,000 iterations. Ten separate sensitivity runs are completed for each combination of population size (4, 8, 12, 16, 20, 32, 40, 48, 56, 64, 72, 80, 88, and 96) and mutation rate (0.001, 0.005, 0.01, 0.05, 0.075, 0.1, 0.12, 0.125, 0.15, 0.175, 0.2, 0.225, and 0.25). The number of function evaluations for each combination of population size and mutation rate of those 10 runs are then averaged to produce plots (Fig. 5.12 for the binary GA results and Fig. 5.13 for the continuous variable GA) of average performance as a function of population size and mutation rate. Both plots contour the logarithm (base 10) of the number of function calls. We see that for both the binary and continuous GA, using too small of a mutation rate prolongs the calculation.

Figure 5.12 plots the number of function evaluations required to solve equation (5.5) using a binary GA. Note that using the smallest mutation rates requires more calls to the cost function to solve the problem. This observation implies that a sufficient number of mutations is required to push the population into the global solution basin for this highly oscillatory cost function. The fewest number of function evaluations are required when the population size is relatively small ($\leq$32) and the mutation rate is moderately large (0.075 to 0.25). For this problem, mutation is a critical operator that keeps the solution from prematurely converging toward the wrong local minimum. The "best" combination for this problem using the binary GA is a population size of 8 and mutation rate of 0.15. The results for the continuous GA appear in Fig. 5.13. In this case, in addition to small mutation rates causing too many function evaluations, too small a population ($\leq$20) also results in a very slow convergence. The "best" combination for the continuous variable GA applied to the Besin problem is a population size of 12 and mutation rate of 0.25.

Our prior work confirms this finding that using relatively small population sizes in combination with high mutation rates is often effective for minimizing the number of function evaluations (Haupt and Haupt 1998, 2000, 2004). One should be careful, however, to note that this conclusion is problem dependent.

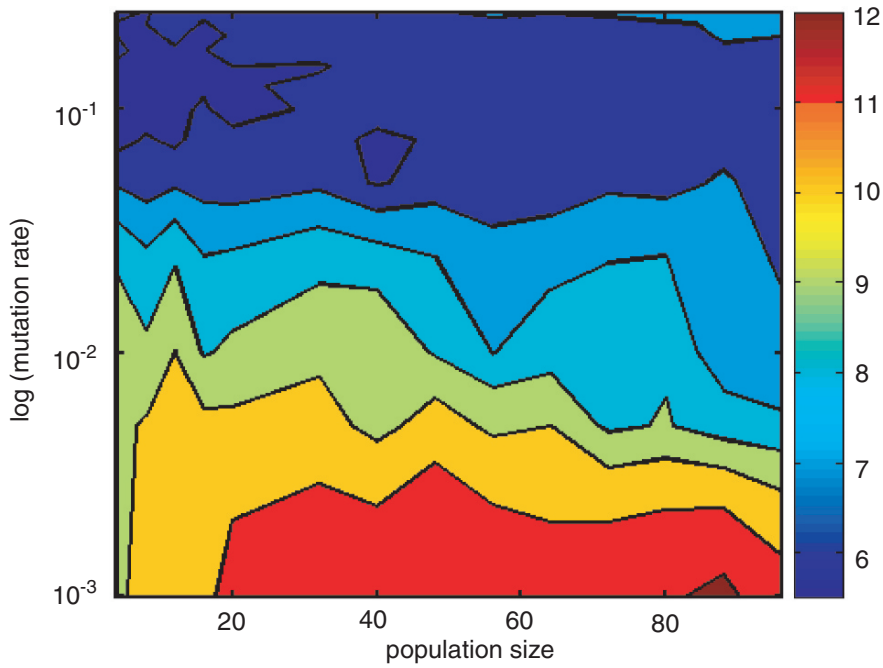


Fig. 5.12 Average number of cost function evaluations over 10 sensitivity runs required to find the solution for a binary GA (log of the number of cost function evaluations)

When there are a large number of unknowns and the cost function has fewer local minima, larger population sizes are sometimes more efficient. We still find, though, that the mutation rate must be sufficiently large.

5.5.6 When to Use a GA

How does the GA compare in speed to other methods? When do we choose to use a GA on our optimization problem rather than a more traditional technique? In

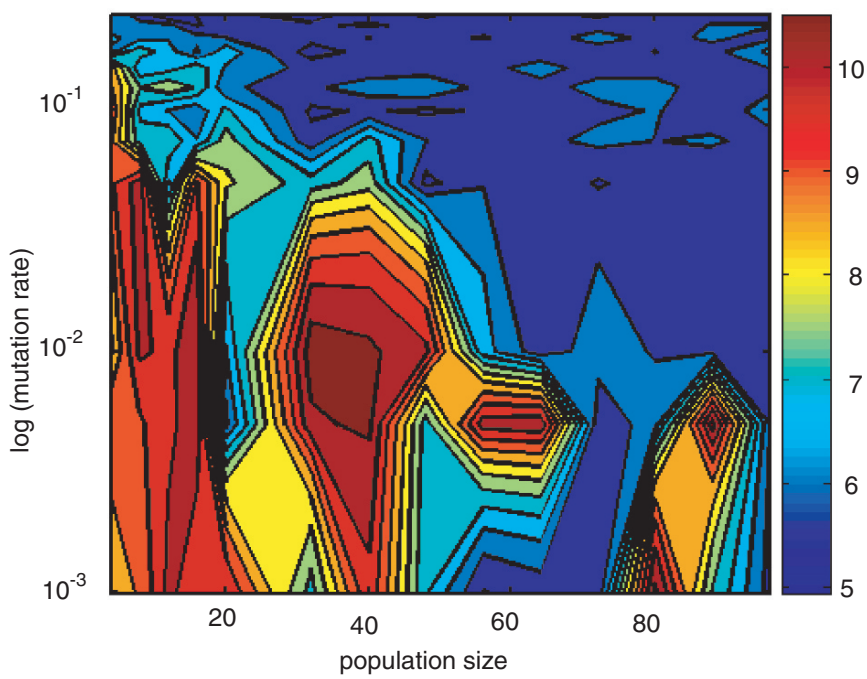


Fig. 5.13 Average number of cost function evaluations over 10 sensitivity runs required to find the solution for a continuous GA (log of the number of cost function evaluations)

general, if we have a well behaved bowl shaped continuously differentiable function and we wish to find the minimum, the gradient descent methods can't be beat. Such methods were designed for those cases and work well there. The GA will not match their speed. In contrast, if we have a very complicated function with lots of local optima, the gradient algorithms will typically find the nearest minimum, which is often local, not global. The usual situation for the practicing scientist or engineer is that he or she configures a problem in a rather large solution space and isn't quite sure what the function looks like in solution space. In that case, it is often wise to try the gradient algorithms first using various different first guesses. If the algorithm finds a different solution for each initialization, the cost function likely has local minima and that is what the algorithm is finding. Then the practitioner knows that using a more robust technique is merited. Those cases are where the GA shines.

Many optimization experts prefer to combine the strengths of the various techniques on their difficult problems. One strategy is to use a hybrid GA; that is, to begin the solution process with a GA until the correct solution basin is discovered, then switch to a fast gradient descent method. One strategy that this author uses is to employ the GA for a specified number of iterations or until a plateau in the convergence plot is reached, then switch to a descent technique. This strategy is often successful at using the GA to determine the basin of attraction then using the ability of the gradient descent method to zoom to the bottom of that basin rapidly. Chapter 14 demonstrates this strategy on a difficult air pollution problem.

5.5.7 Genetic Algorithms on a Parallel Computer

Everything we have said thus far about the speed expected of a GA assumes that we are using a serial computer. There are ways to speed GAs on a parallel computer that (1) make them competitive with other techniques in speed, (2) make efficient use of the processors, and (3) may even tune the GA to produce global minima more quickly. It is beyond our scope to go into too much detail here, but we do want to point out that for some problems, creating co-evolving populations not only makes the algorithm amenable to distributing among processors, but it also helps discover a more global minimum and speeds the convergence, even if implemented on a single processor machine. The many brands of parallel GAs can be characterized grossly into three primary categories: master-slave, island GA, and cellular GA. Figure 5.14 is a graphical depiction of these three parallel implementations.

The master-slave implementation of the GA is the most similar to the serial GAs that we have been discussing: they merely distribute the cost function evaluations to be done by slave processors that report their results to the master processor. This method is easy to implement and no subpopulations are required.

The island GA allows simultaneous evolution of independent populations. There is often a prescription for some periodic migration between these subpopulations to increase diversity. Although this method is a bit more difficult to implement, it is often successful at increasing the diversity of the overall population while speeding evolution of the individual populations through independent evolution that doesn't need to wait for communication from any master.

The cellular implementation of parallelism is ideal for architectures where communication with nearest neighbor nodes is faster than for more distant processors. In this case, each individual often evolves on its own node with options of mating only with its nearest neighbors. More detail on parallel GAs can be found in Gordon and Whitley (1993), Alga and Tomassini (2002), and Haupt and Haupt (2004).

5.5.8 Cost Function Construction

Finally, we would be negligent if we did not emphasize the importance of wise construction of the cost function. This element is, of course, unique to the problem at hand. The cost function is often the determining factor for both how much CPU time is required as well as how quickly the GA converges to the solution. A few of the typical general programming tips hold especially true for the GA since the cost function is called repeatedly – vectorize the code where possible (taking advantage of the storage order and special vectorization tools of the language being used) and

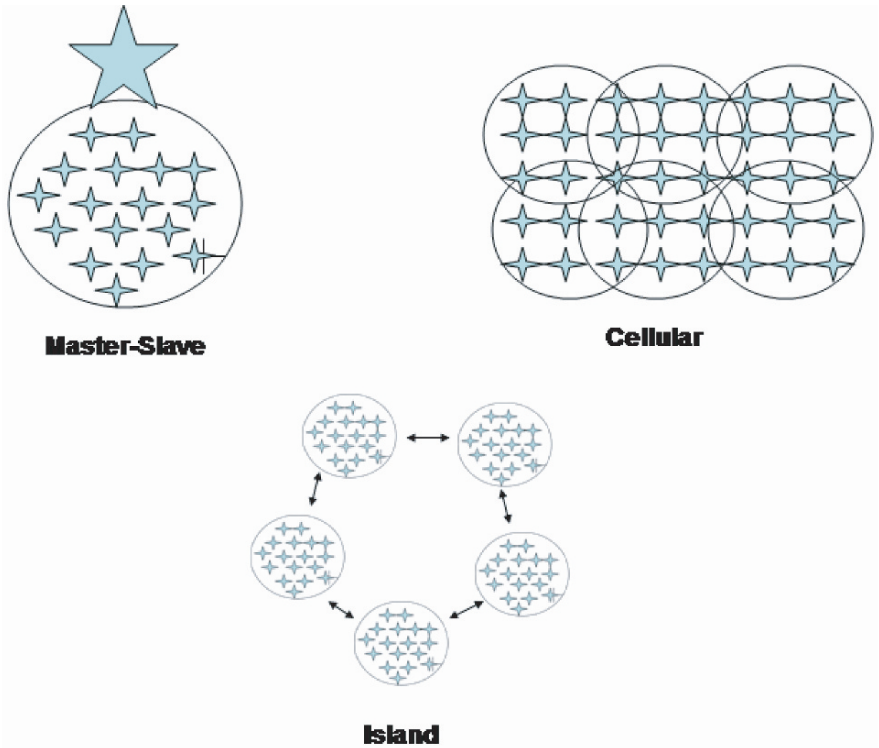


Fig. 5.14 Schematic of the three general types of parallel GA configurations

avoid using constructs known to slow down the code (such as contingencies). In addition, the GA allows more creative design of cost functions than many traditional techniques. For instance, many traditional problems minimize the L_2 norm (sum of the squares) of some difference from a known or previous value. With a GA, it is easy to explore using other powers of the difference – in some cases an L_1 norm (absolute value of the differences) is preferable while in others, higher powers are useful if we want to weight the technique to avoid outliers. In yet other problems, least squares need not play any role. For instance, in developing contingency tables for occurrences of a meteorological condition (e.g. predicting whether or not it will hail), the models are traditionally built using a least squares methodology, but then judged using more complex metrics such as Fraction Correct, Critical Success Index, or Heidke Skill Scores. The GA is capable of training the model using those same metrics to optimize agreement with past data (Marzban and Haupt 2005 and Chapter 18 of this volume). Therefore, one additional strength of using a GA to optimize a function is the greater freedom in designing the function to be optimized. In addition, it may be appropriate to weight different portions of a cost function differently. We did this for the initial guppy problem (Section 5.2) when balancing the competing parameters of the attractiveness to mates versus the likelihood of being eaten by a predator.[3] Finally, in some cases, constraints can be built directly into cost functions to avoid specific portions of parameter space. If such a constraint is necessary, one could simply add a term to the cost function to increase the cost if the function value strays too far from that expected – i.e. impose a penalty for straying too far in parameter space.

5.6 Application to a Dynamical Systems Problem

Let's revisit our initial problem of optimizing a population of guppies to survive in a particular

[3] An alternative approach would be to map the pareto front of the competing parameters. That technique is beyond the scope of this chapter but is covered in Haupt and Haupt (2004).

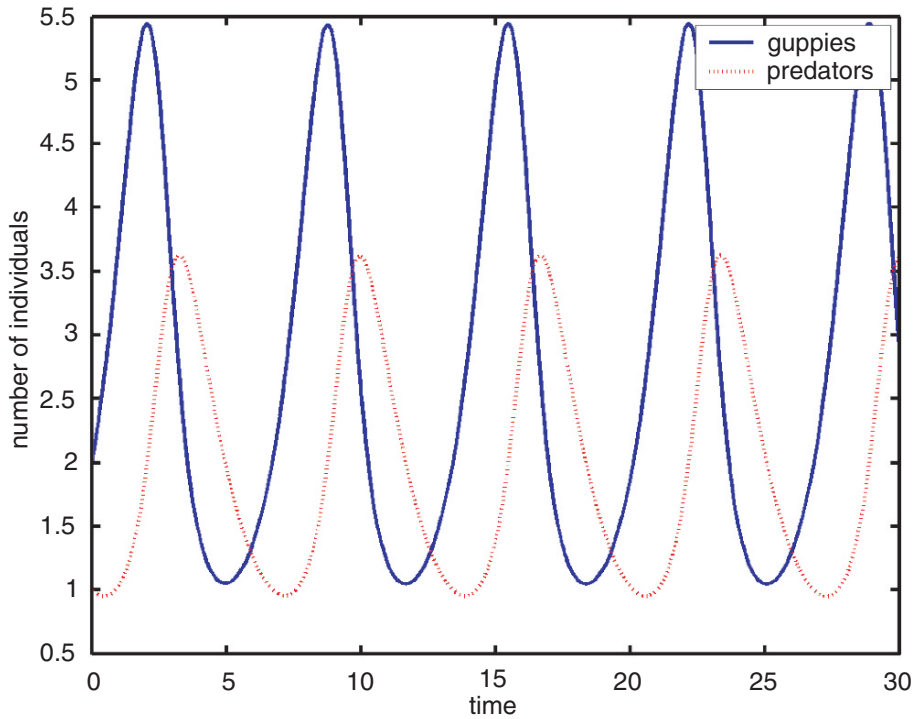


Fig. 5.15 Time evolution of the predator-prey equations (5.15)

environment populated by a predator. In Section 5.2 we directly optimized the guppy population using a binary value GA. Modern population biologists often instead model population dynamics with differential equations. For instance, the classic predator-prey problem, also known as the Lotka-Volterra equations (see Chapra and Canale 1998 for a brief discussion of these equations) is formulated as:

$$\begin{aligned} \frac{dx}{dt} &= ax - bxy \\ \frac{dy}{dt} &= -cy + dxy \end{aligned} \tag{5.15}$$

where

x = number of prey (guppies)
y = number of predators
a = guppy birth rate
c = predator death rate
b,d = interaction coefficients

This is a nonlinear system with coupling of the rate equations for the change in the number of prey and predators. It is reasonably trivial to integrate this equation in time and to characterize it in phase space. Setting the parameters $a = 1.2$, $b = 0.6$, $c = 0.8$, and $d = 0.3$ and doing a Runge-Kutta time integration with a time step of 0.01 produces a time series as shown in Fig. 5.15. Figure 5.16 is a plot in the phase space of guppies versus predators. The egg shaped pattern denotes a limit cycle that is repeated indefinitely. The nonlinear coupling of these equations is what makes this limit cycle occur.

What if we had measured data of the time evolution of the guppies and their predators but didn't know the functional form? What if we wished to come up with a model for the rate of change of guppies and predators given this data that we had obtained? For the moment, let's assume that this time series we generated through integrating equations (5.15) is that data. We wish to use these data to reproduce the time dependent behavior of the guppy/predator interaction.

The first step in solving such a problem is to generate a likely form for the solution. The simplest approach is to assume a linear model, such as:

$$\mathbf{s}_t = \mathbf{L}\mathbf{s} + \mathbf{C} \tag{5.16}$$

where $\mathbf{s}$ is the vector of variables (x, y), the subscript t denotes its time variation, $\mathbf{L}$ is the linear matrix

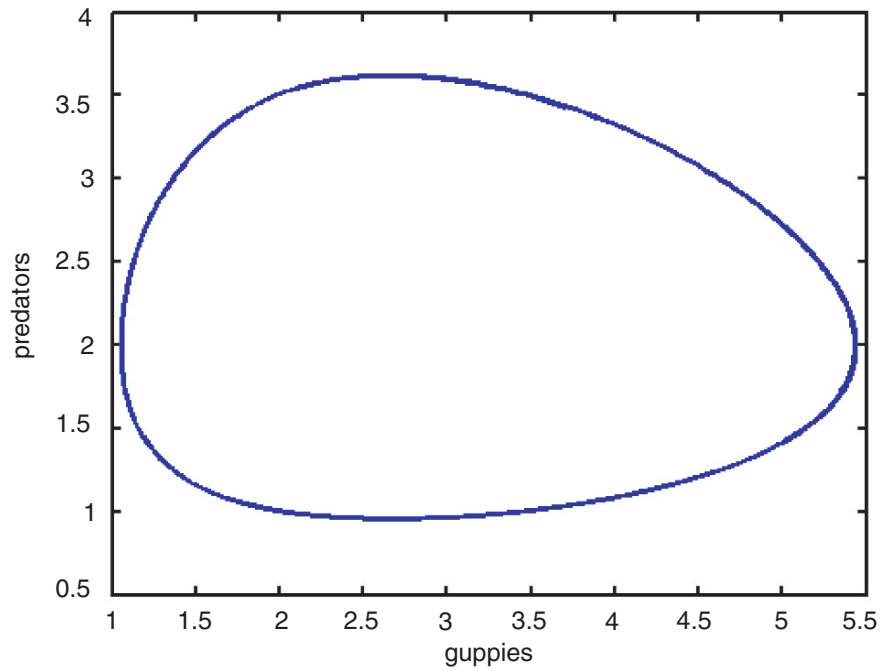


Fig. 5.16 Phase space plot of guppies vs. predators for equations (5.15)

operator, and **C** is a constant matrix. One doesn't need an optimization algorithm to fit data to this model: just use standard least squares parameter estimation to determine the unknown elements of matrices **L** and **C**. Doing that and using equation (5.16) to integrate it forward in time produces the time series observed in Fig. 5.17. We see here that this linear model finds a solution of one predator and a monotonically growing number of guppies in time. The corresponding phase space plot appears as Fig. 5.18. It is obvious that this diverging solution is not a very good match to the actual solution of Figs. 5.15 and 5.16.

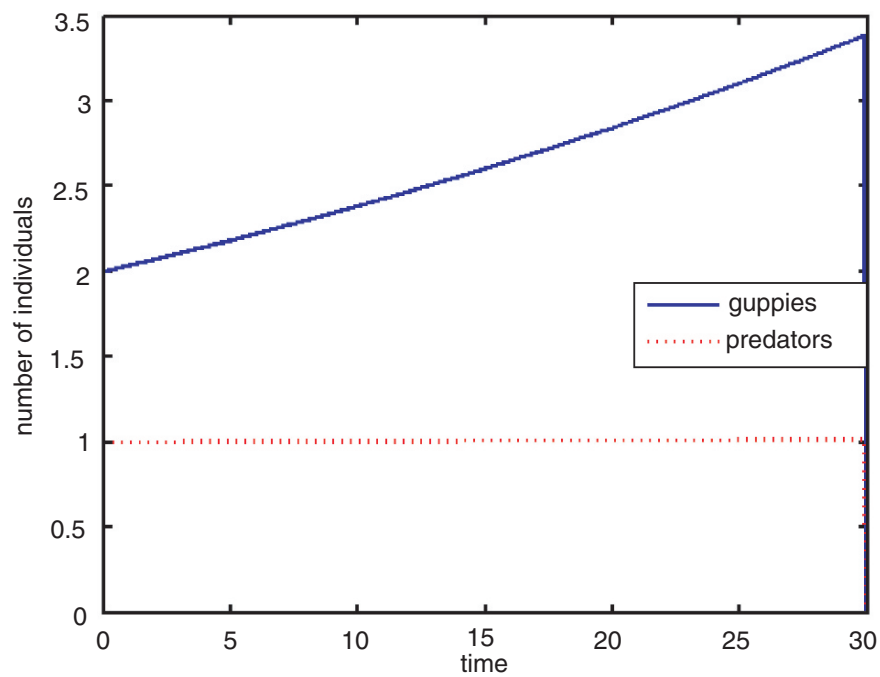


Fig. 5.17 Time series of linear empirical model fit (equation 5.16) to the Lotka-Volterra equations

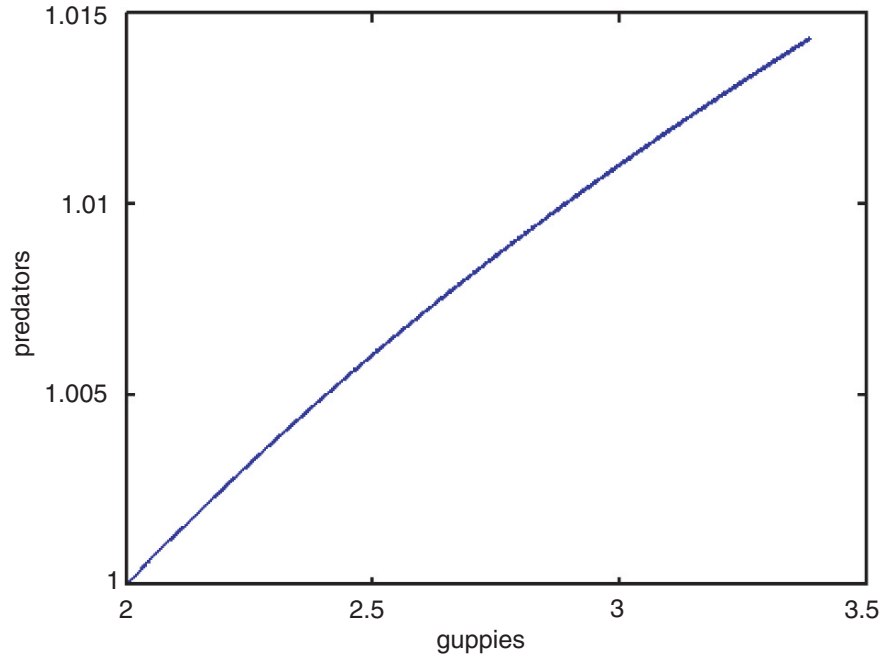


Fig. 5.18 Phase space plot of linear empirical model fit (equation 5.16) to the Lotka-Volterra equations

The next approach is to try configuring a nonlinear model of the guppy/predator time behavior. We conjecture (given that we know the solution) that the nonlinearity is quadratic. Therefore, the simplest time dependent model that includes both a linear and quadratic term is:

$$\mathbf{s_t} = \mathbf{N}\mathbf{s}^{\mathbf{T}}\mathbf{s} + \mathbf{L}\mathbf{s} + \mathbf{C} \tag{5.17}$$

This nonlinear equation can not be as simply solved using least squares techniques. An equation to do that

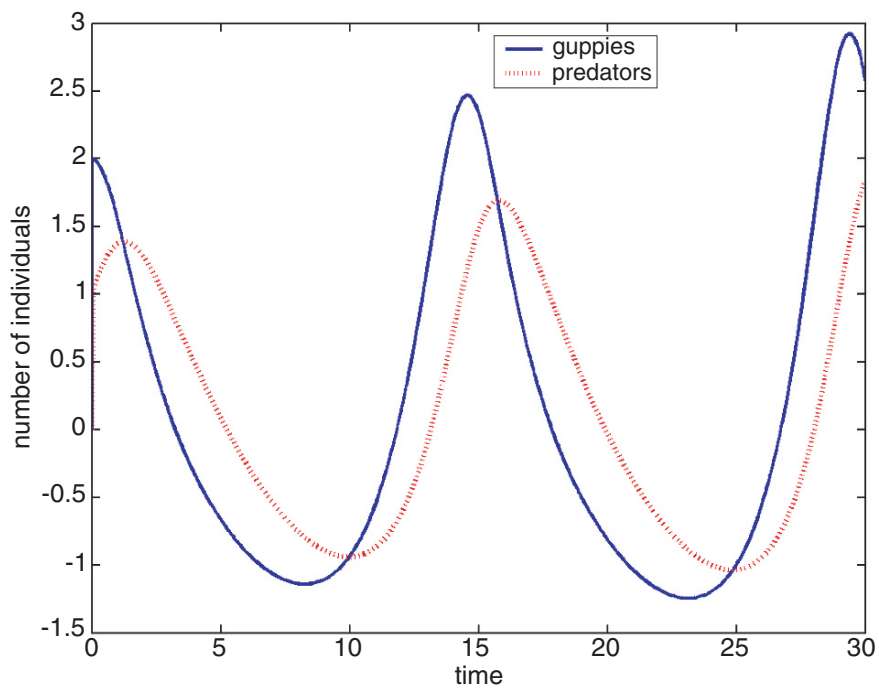


Fig. 5.19 Time series of GA solved nonlinear empirical model fit (equation 5.17) to the Lotka-Volterra equations

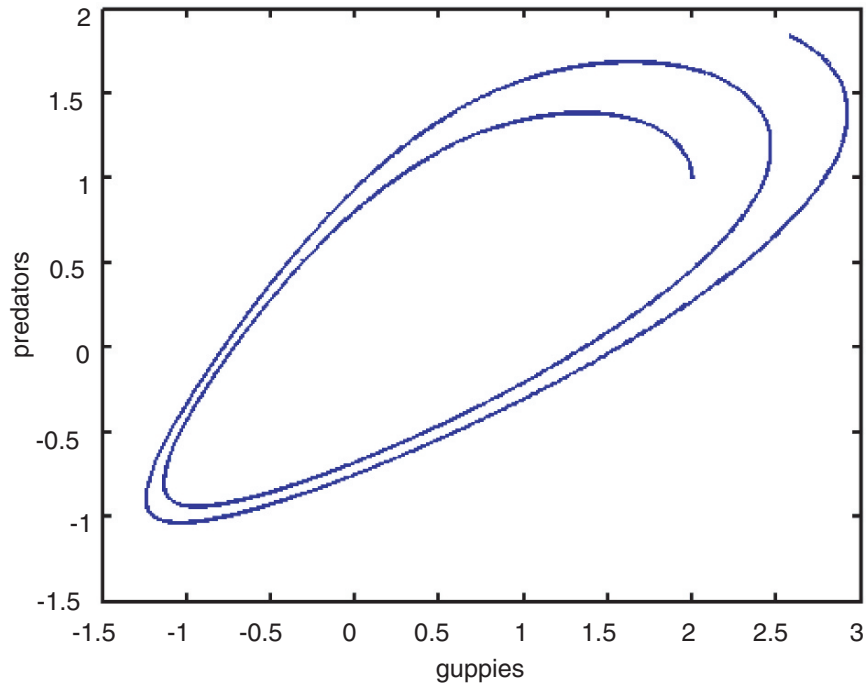


Fig. 5.20 Phase space plot of GA solved nonlinear empirical model fit (equation 5.17) to the Lotka-Volterra equations

can be found (Haupt 2006), but its solution involves inverting a fourth order tensor, which is not a trivial problem. Instead, we choose to configure this problem as one in optimization where we seek to minimize:

$$\text{cost} = \mathbf{s_t} - \left(\mathbf{N}\mathbf{s}^{\mathbf{T}}\mathbf{s} + \mathbf{L}\mathbf{s} + \mathbf{C}\right) \tag{5.18}$$

Given that we have the data of $\mathbf{s}$ and its tendency, $\mathbf{s_t}$, we can then directly solve for the elements to the matrices $\mathbf{N}$, $\mathbf{L}$, and $\mathbf{C}$. In addition, we use information on the symmetries and relations expected between these matrices to minimize the number of real variables that we need to find (Haupt 2006). A GA is used to find

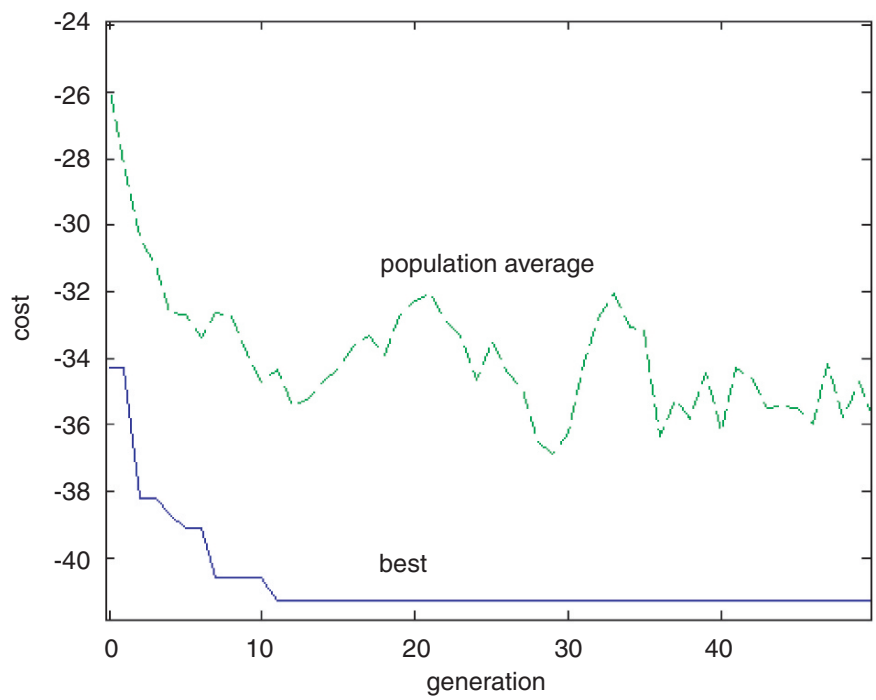


Fig. 5.21 GA convergence for the nonlinear guppy/predator empirical model

the solution – the time tendency appears as Fig. 5.19 and phase space plot as Fig. 5.20. The match to the original data (Figs. 5.15 and 5.16) is not perfect, but the general shape and oscillatory nature of the behavior is obtained. Figure 5.21 is the GA convergence curve. We see that it took very few iterations (10 for this run) to converge to the optimal solution. Note that reproducing the exact balance for a nonlinear model with an empirical model is extremely difficult and it is amazing that we have come this close with an optimization algorithm. Chapter 18 further develops this technique in the context of a chaotic dynamical system.

5.7 Conclusions

This chapter strives to give a basic introduction to genetic algorithms. We saw that GAs can be a useful technique to solve optimization problems. We also saw that they can be used for building models, whether of natural processes such as guppy evolution or of dynamical systems such as the Lotka-Volterra equations. We have only begun to scratch the surface of potential GA applications or of how to best apply the GA. There are entire books on what GAs are and how to apply them (Holland 1975; Goldberg 1989; Davis 1991; Mitchell 1996; Haupt and Haupt 1998, 2004). Chapter 18 gives a smattering of examples of how to configure a problem for a GA and reviews prior use of GAs in the environmental sciences. Chapter 14 delves into detail on how a GA was used in a specific problem in source characterization of a source of air contaminant and how that problem could be reconfigured due to GA robustness to additionally solve for meteorological variables.

The GA is a useful tool, but it is not always the first tool to try on an optimization problem. As stated in Section 5.5, traditional methods may be more time efficient on easy-to-solve problems. The strength of the GA is in optimizing difficult problems with lots of local minima. A carefully configured GA is one of the most robust tools for finding such solutions. It may not be as fast, but it is amazingly successful at identifying the basin of the global minimum. As will be demonstrated in Chapter 14, a hybrid approach of using the GA to that point, then switching to a gradient descent method is often the quickest way to a difficult solution.

The hope of this author is that we have provided the reader with enough of a view of GAs to capture his or her interest and have helped motivate the reader to apply it to his or her own problems.

Acknowledgements The GA used here was jointly authored with Randy L. Haupt. The genetics aspects of the guppy evolution problem were brought to my attention by Amy J. Haupt and her biology teacher, Beth Paterson. Bonny A. Haupt programmed the cost functions for the guppy evolution problem.

References

Alga, E., & Tomassini, M. (2002). Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, *6*, 443–462.

Chapra, S. C., & Canale, R. P. (1998). *Numerical methods for engineers* (3rd ed.). Boston: McGraw-Hill.

Davis, L. (Ed.) (1991). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.

De Jong, K. A. (1975). *Analysis of the behavior of a class of genetic adaptive systems*. Ph.D. dissertation, The University of Michigan, Ann Arbor, MI.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. New York: Addison-Wesley.

Gordon, V. S. & Whitley, D. (1993). Serial and parallel genetic algorithms as function of optimizers. In S. Forrest (Ed.), *ICGA-90: 5th international conference on genetic algorithms* (pp. 177–183). Los Altos, CA: Morgan Kaufmann.

Haupt, R. (2007). Antenna design with a mixed integer genetic algorithm. *IEEE Transactions on Antennas and Propagation*, *55*(3), 577–582.

Haupt, R. L., & Haupt, S. E. (1998). *Practical genetic algorithms* (177 pp.). New York: Wiley.

Haupt, R. L., & Haupt, S. E. (2000). Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors. *Applied Computational Electromagnetics Society Journal*, *15*(2), 94–102.

Haupt, R. L., & Haupt, S. E. (2004). *Practical genetic algorithms*, *second edition with CD* (255 pp.). New York: Wiley.

Haupt, S. E. (2006). Nonlinear empirical models of dynamical systems. *Computers and Mathematics with Applications*, *51*, 431–440.

Haupt, S. E., Haupt, R. L., & Young, G. S. (2008). A mixed integer genetic algorithm used in chem-bio defense applications, accepted by *Journal of Soft Computing*.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: The University of Michigan Press.

Marzban, C., & Haupt, S. E. (2005). On genetic algorithms and discrete performance measures. *AMS 4th Conference on Artificial Intelligence*, San Diego, CA, paper 1.1.

Mitchell, M. (1996). *An introduction to genetic algorithms*. Cambridge, MA: MIT Press.

Introduction to Fuzzy Logic

6

John K. Williams

6.1 Introduction

In the early 1990s, scientists at the National Center for Atmospheric Research were asked to help the Federal Aviation Administration objectively determine which flight service stations throughout the United States handled the most hazardous weather conditions, and hence should be spared from congressional budget cuts. They had access to 15 years of meteorological data from each location, including winds, temperature, fog, rain, and snow at 1-min intervals, as well as information about the air traffic density and a number of other factors. However, the level of aviation hazard was not indicated by any single statistic, but by the nature, frequency and duration of the conditions and their combinations. How could the stations be ranked in a reasonable, objective way?

The scientists began by surveying a group of subject domain experts – pilots, meteorologists, and airline dispatchers – quizzing them on what factors, or combinations of factors, they considered most dangerous. Using the results of these surveys along with the repository of historical data, they then computed a hazard score for each flight service station, and ranked them. When the final report was presented to the group of experts, they opened it and began to laugh – everyone agreed that the stations with the most hazardous weather were at the top of the list. Unwittingly, the NCAR scientists had created a fuzzy logic algorithm, efficiently encoding the experts' knowledge in a set of rules that reproduced their approach to assessing the level of hazard presented by each unique set of weather conditions.

The purpose of artificial intelligence algorithms, broadly stated, is to perform tasks in a way that mimics human intelligence. Many of the approaches to creating artificial intelligence algorithms presented in this book are *machine learning* algorithms that automatically recognize patterns or functional relationships in data. By training on a historical dataset or incrementally adapting as new examples are gathered, these approaches mimic a human's ability to draw inferences from data. Of course, a machine doesn't truly "learn" as a human does; rather, it uncovers statistical relationships. To achieve meaningful results, a human expert is still required to select an appropriate machine learning technique for a given problem, determine a representative data set, design data preprocessing or feature representation, set the evaluation criteria, and experiment with algorithm parameters to obtain optimal learning behavior.

Fuzzy logic presents an alternative approach to artificial intelligence by providing a framework for imitating a human expert's approach to solving a particular problem. This approach falls under the class of artificial intelligence algorithms called "expert systems" that encode expert knowledge as a set of heuristics, or rules. For instance, a medical expert system might help diagnose a disease based on a patient's answers to a series of questions. With each answer, the number of likely possibilities would diminish until finally the most probable diagnosis was determined. A key motivation for developing fuzzy logic expert systems is the recognition that human experts rarely make decisions based on a sequence of well-defined logical

John K. Williams (✉)
Research Applications Laboratory
National Center for Atmospheric Research
P.O. Box 3000, Boulder, CO 80307, USA
Phone: 303-497-2822; Fax: 303-497-8401;
Email: jkwillia@ucar.edu

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

statements. For instance, many questions that might be asked by a doctor diagnosing an illness may not have clear or definitive answers, and even the questions themselves might be ambiguous. Rather, humans often draw inferences from a preponderance of the evidence available to them – even if it is incomplete – with each source of information weighed according to its reliability and the strength of its connection to the outcome being evaluated.

A distinctive contribution of fuzzy logic is that it provides a conceptual framework for representing natural language concepts and reasoning by expanding on the traditional notion of a *set*, or collection of objects. For instance, the noun "chair" is in a sense equivalent to the set of objects identified as chairs. The set of green chairs is the *intersection* of the set of objects that are chairs and objects that are green. A proposition like "all chairs have legs" is a claim that the set of chairs is a *subset* of the set of objects having legs. The "fuzzy" part of fuzzy logic arises from a recognition that the sets of objects referred to by words are not "crisp" like the sets of traditional mathematics in which a given object is either in a particular set or outside it. Rather, in everyday language – and, by extension, in human reasoning – concepts often have "fuzzy" boundaries. For instance, take the concept of being a chair. Do thrones, or benches, or bean bags, or step stools, or tree stumps count as "chairs"? In traditional logic, each of these objects is either a chair or it isn't. In fuzzy logic, on the other hand, set membership is allowed to be partial. Thus, on a scale of 0 to 1, we might say that a throne is a chair to the degree 0.9, while a tree stump might be a chair to the degree 0.1. The proposition "all chairs have legs" can then be the understood as applying to different objects only to the degree that they are members in the set of "chairs". Reasoning with fuzzy logic accommodates and even exploits ambiguity, a feature that turns out to be immensely powerful for expert systems.

By formally accommodating the ambiguity of natural language, fuzzy logic makes it possible to encode human knowledge into relationships between concepts represented as fuzzy sets. Reasoning involves logical manipulation of these statements, and multiple lines of reasoning may be aggregated to form a conclusion. Thus, fuzzy logic can be viewed as the extension of classical logic to fuzzy sets and their manipulation. In addition, in many research communities the term "fuzzy logic" has come to be applied more broadly to the theory and practice of computing with fuzzy sets, or indeed any expert system or set of heuristics that preserves the uncertainty or ambiguity in data until a final "decision point." Fuzzy logic systems have become increasingly popular, both in industrial and environmental science applications, because they use information efficiently, are relatively simple to design and implement, and often perform impressively well. They do not require training data, models, or conditional probability distributions; they are robust to uncertain, missing and corrupted data; they naturally constrain the possible output to "reasonable" values; and they often have a "common sense" structure that makes them relatively easy to interpret and modify. While they may not always produce the optimal solution, fuzzy logic algorithms fall within the increasingly popular domain of "soft computing" methods that produce very good, practical and relatively inexpensive solutions to many problems.

This chapter presents a brief history of the development of fuzzy logic; describes fuzzy sets, fuzzy set theory, fuzzy numbers and fuzzy logical operations; discusses two types of fuzzy inference and a couple of variants; presents a fuzzy clustering method; and explains how these pieces can be used to create fuzzy logic algorithms that may be "tuned" using empirical data if desired. The field of fuzzy logic theory and techniques is very broad, though, and this short chapter is only able to introduce a brief overview of elements that have been found most useful to the author. Interested readers are encouraged to consult the texts by Chi et al. (1996), Klir and Folger (1988), McNiell and Freiberger (1993), Tanaka (1996), or Zimmerman (1996) and the collections of original papers in Klir and Yuan (1996) or Yager et al. (1987) for a more thorough exposition.

6.2 A Brief History

Fuzzy logic owes its existence to Lotfi Zadeh, who in 1965 published an article entitled "Fuzzy Sets" in a journal he edited, *Information and Control*, after having it rejected from several other journals (Zadeh 1965). Faced with skepticism from his contemporaries, Zadeh struggled for years to gain the recognition that his revolutionary ideas deserved. However, some researchers were quick to see the practical

utility of his work. In 1973, Ebrahim Mamdami and Sedrak Assilian successfully used fuzzy logic to build a controller for a steam engine. In 1978, Laritz Peter Holmblad and Jens-Jorgen Østergaard developed and commercialized a fuzzy logic controller for industrial cement kilns. And in 1979, Hans Berliner's BKG 9.8 program beat the backgammon world champion by the convincing score of 7-1. The use of fuzzy logic developed particularly rapidly in Japan, where a subway system developed by Hitachi using predictive fuzzy controllers entered service in Sendai in 1987. These early successes were followed by an explosion of applications in consumer electronics and appliances, finance, and industrial process control, first in Asia and eventually also in Europe and the United States.

The value of fuzzy logic algorithms in the environmental sciences gained recognition beginning in the early 1990s. In 1993, MIT Lincoln Laboratory developed a fuzzy logic algorithm for detecting gust fronts from Doppler radar data (Delanoy and Troxel 1993). In 1994, researchers at the National Center for Atmospheric Research created an algorithm for detecting microbursts – wind events associated with thunderstorms that had been responsible for a number of airplane crashes – based on Doppler radar data (Albo 1994, 1996; see also Merritt 1991). Fuzzy logic was soon employed in algorithms for processing sensor data, recognizing hazardous weather conditions, producing short-range forecasts, and providing weather decision support information to aviation, state departments of transportation, and other users. By the early 21st century, a large number of the decision support systems produced by NCAR's Research Applications Laboratory incorporated fuzzy logic in one way or another. Some of these applications are described in detail in Chapter 17.

6.3 Classical and Fuzzy Sets

A fundamental concept of fuzzy logic is that of the fuzzy set. As mentioned earlier, a fuzzy set is an extension of the classical notion of a set: whereas a classical set divides the universe of objects into two distinct categories, those in the set and those outside it, a fuzzy set permits intermediate degrees of membership. More formally, every classical set is determined by its *characteristic function*, a mapping that assigns every object in the set a membership value of 1, and every object outside it a value of 0. For instance, the set of "tall people" might have a characteristic function that assigns 0 to all people having heights less than 5′ 9″ and 1 to those with height 5′ 9″ or greater. The equivalent concept for fuzzy sets is that of a *membership function*, which assigns every object a value between 0 and 1 representing its *degree* of membership in a set. The membership function for the fuzzy set of "tall people" might be 0 for people less than 5′ 4″ tall, then rise gradually to 0.5 for people 5′ 9″ tall, and then continue to rise to a value of 1 for people 6′ 3″ tall or taller. Most of us would agree that this representation more accurately represents the concept of being a "tall person": rather than enforcing a discontinuous cutoff at some particular height, the fuzzy set quantifies the ambiguity (Fig. 6.1).

The reader may have noticed that the concept of fuzzy membership is reminiscent of probability theory,

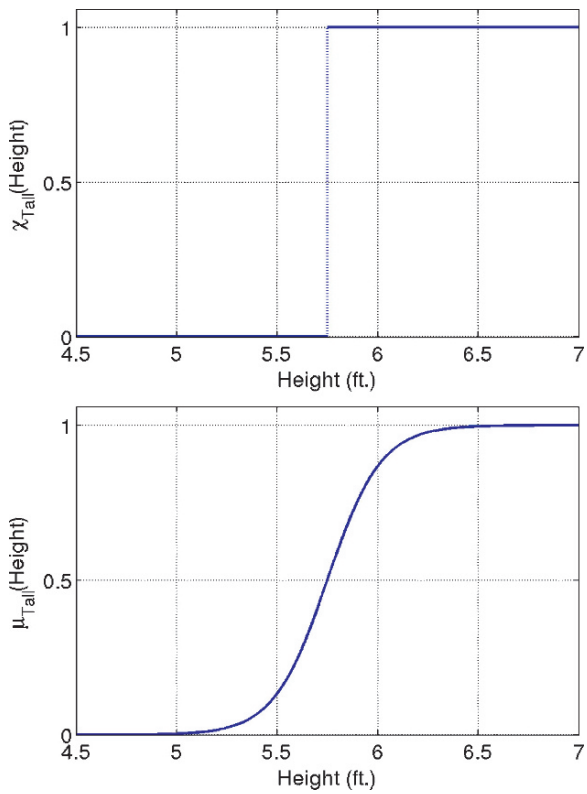


Fig. 6.1 (Top) A classical set characteristic function (0 or 1) for "tall" as a function of a person's height. (Bottom) A fuzzy set membership function representing continuous "degrees of tallness" as a function of height.

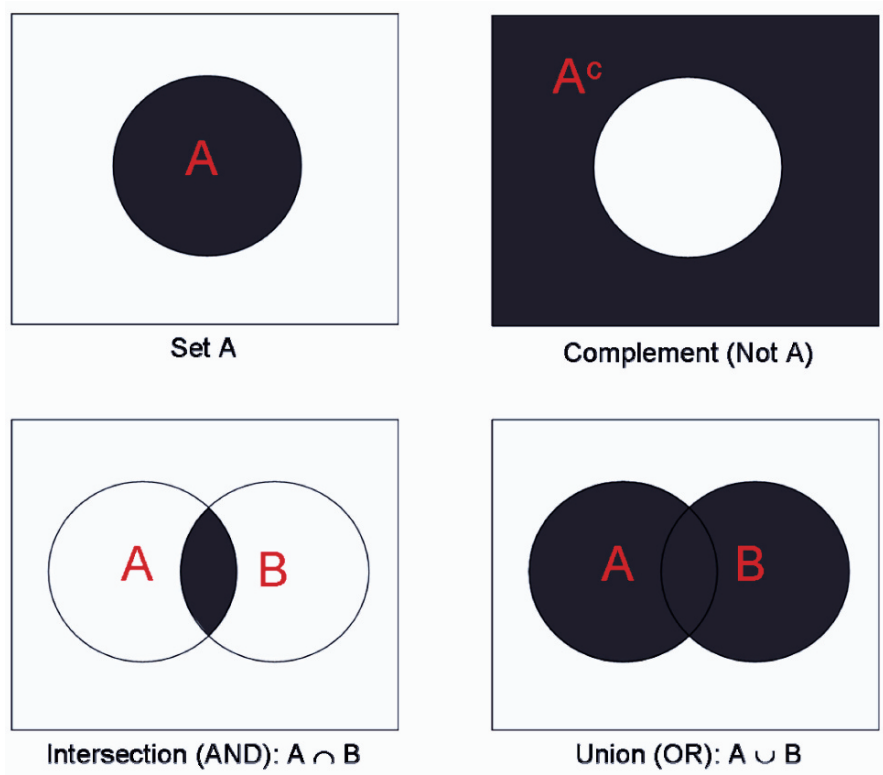


Fig. 6.2 Depiction of the classical set operations complement, intersection and union. Black shading represents elements in the resultant set.

in which "events" are assigned values between 0 and 1 representing their likelihood. Both fuzzy memberships and probabilities can be interpreted as "measures" and manipulated mathematically. However, probabilities and fuzzy set memberships are really quite different concepts. While probabilities capture summary information about random events – for instance, the likelihood that a person chosen randomly from a crowd will have a height above 5′ 9″ – a fuzzy membership captures the ambiguity in a concept itself, e.g., what we mean by the very notion of "being tall".

Classical set theory defines several important operations on sets that are useful in formal logic. For instance, the *complement* of a set A, denoted A^c, is the set of all objects not in A (Fig. 6.2). The characteristic function for A^c is the binary opposite of that for A; that is, when the characteristic function for A is 1, the characteristic function for A^c is 0, and vice versa. Writing the characteristic function for A in the traditional fashion as χ_A (here χ is the Greek letter "Chi") and that of A^c as χ_{A^c}, this relationship can be expressed in an equation as $\chi_{A^c} = 1 - \chi_A$. (Note that this is a *functional* equation, that is, one that involves a relationship between functions rather than numbers. It is equivalent to saying that the functions χ_{A^c} and χ_A share the same domain, and that $\chi_{A^c}(a) = 1 - \chi_A(a)$ for all elements a in that domain.) As an example, suppose A is the classical set of "tall people" as defined in the previous paragraph, and suppose Peter is 5′ 8″. Then $\chi_A(\text{Peter}) = 0$ and $\chi_{A^c}(\text{Peter}) = 1 - \chi_A(\text{Peter}) = 1$. The complement of "tall people" is the set of people who are not tall – a set that includes Peter.

In fuzzy set theory, the binary characteristic function is replaced by a continuous *membership function* that can take on any value between 0 and 1 (Fig. 6.3). Writing the membership function for the fuzzy set B as μ_B (μ is the Greek letter "Mu") and that of its fuzzy complement B^c as μ_{B^c}, then $\mu_{B^c} = 1 - \mu_B$. Since fuzzy sets and membership functions are equivalent, this functional equation *defines* the fuzzy set complement. Now suppose that the fuzzy set B of "tall people" is defined so that $\mu_B(\text{Peter}) = 0.4$. Then $\chi_{B^c}(\text{Peter}) = 1 - \chi_B(\text{Peter}) = 1 - 0.4 = 0.6$. In words, one could say that Peter is a member of "tall people" to the degree 0.4, and a member of "not tall people" to degree 0.6. As another

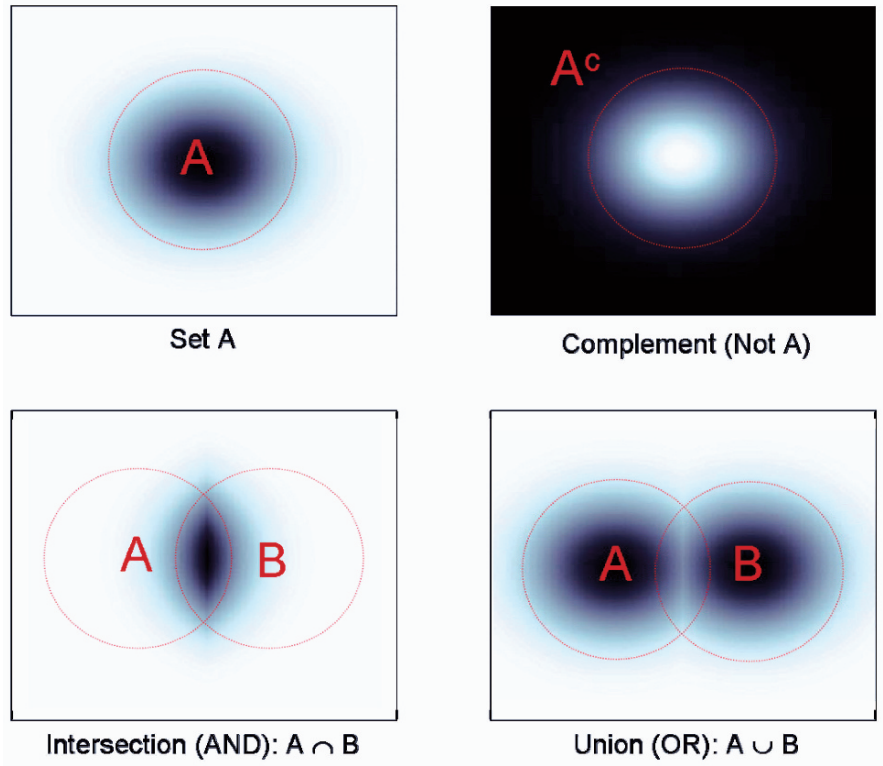
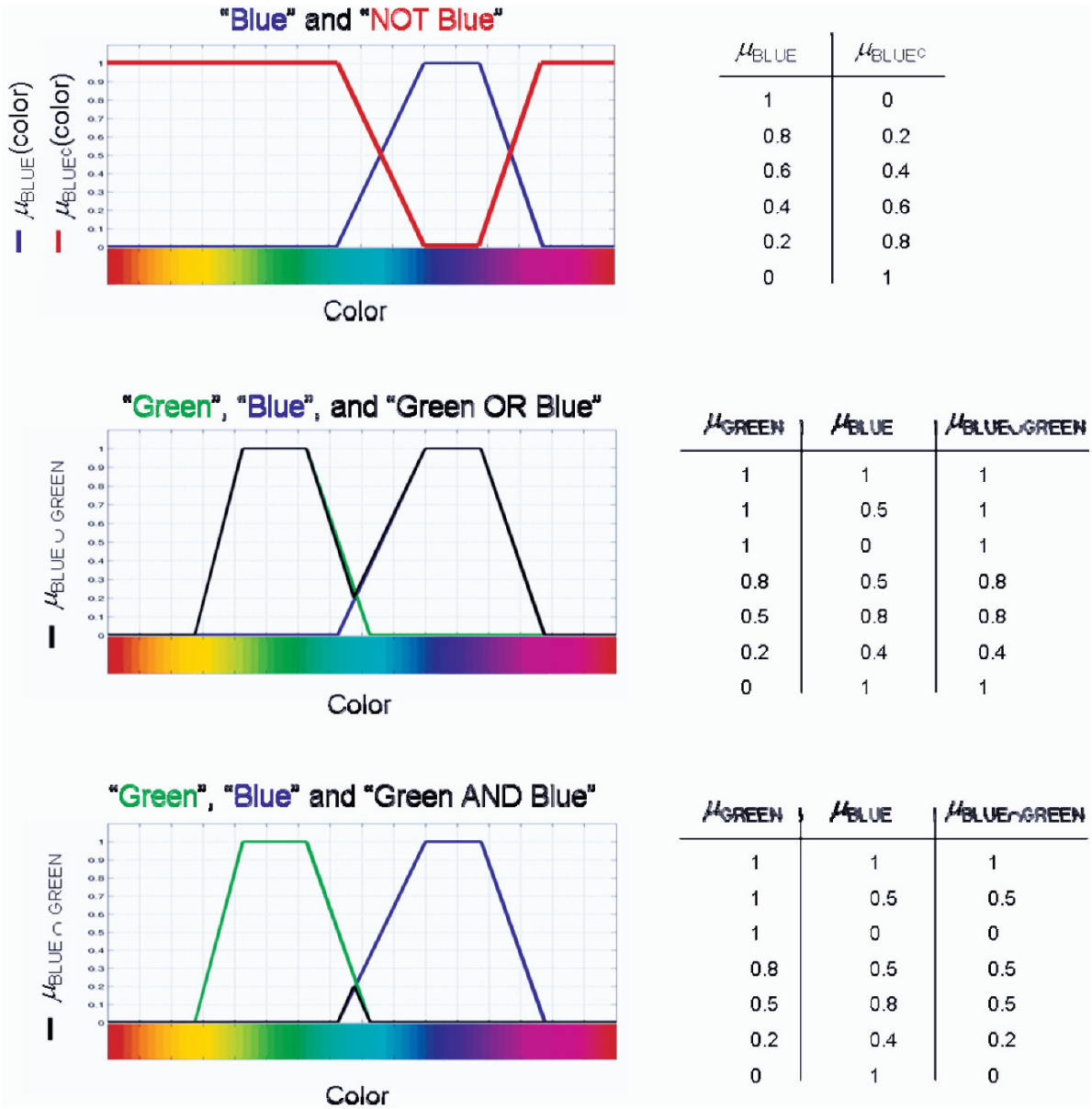


Fig. 6.3 Depictions of fuzzy set operations. The degree of membership in the resultant set is represented by shades of gray, with white points having membership 0 and black points having membership 1. Dotted red circles represent the 0.1 membership contours of the original fuzzy sets.

example, consider colors. When an object is identified as "blue", there is really quite a range of frequency values on the electromagnetic spectrum that could be referred to, ranging from blue-green to almost purple. Figure 6.4 (top) illustrates membership functions for the concept "blue" and its complement, "not blue".

Other elements of classical set theory have similarly natural analogues for fuzzy sets. The *union* of two classical sets A and B, written $A \cup B$, is the set of all elements in either set A or set B, or both. In terms of the characteristic function, $\chi_{A\cup B} = \max(\chi_A, \chi_B)$; that is, $\chi_{A\cup B} = 1$ when either $\chi_A = 1$ or $\chi_B = 1$ (or both). For fuzzy sets A and B, the union is defined by the membership function equation $\mu_{A\cup B} = \max(\mu_A, \mu_B)$, as illustrated using the color concepts "blue" and "green" in Fig. 6.4 (middle). The *intersection* of two classical sets A and B, written $A \cap B$, is the set of all objects in both sets A and set B. In terms of the characteristic function, $\chi_{A\cap B} = \min(\chi_A, \chi_B)$; that is, χ_A and χ_B must both be 1 for $\chi_{A\cap B}$ to be 1. The intersection of two fuzzy sets A and B is defined by the membership function equation $\mu_{A\cap B} = \min(\mu_A, \mu_B)$. This is illustrated in Fig. 6.4 (bottom). For two classical sets A and B, we say that A is a *subset* of B and write $A \subset B$ if all objects (or *elements*) of A are also in B. That is the same as saying that the characteristic function of A can be 1 only when the characteristic function of B is 1, that is, $\chi_A \leq \chi_B$. Similarly, for fuzzy sets A and B, A is a *subset* of B if all elements of A are also in B to at least an equal degree, that is, if $\mu_A \leq \mu_B$ (see Fig. 6.5).

Several consequences of these definitions that hold for classical sets also hold for fuzzy sets. For instance, if $A \subset B$ and $B \subset C$, then $A \subset C$ since $\mu_A \leq \mu_B$ and $\mu_B \leq \mu_C$ implies $\mu_A \leq \mu_C$. And if $A \subset B$ and $B \subset A$, then $A = B$ because $\mu_A \leq \mu_B$ and $\mu_B \leq \mu_A$ implies $\mu_A = \mu_B$. The reader may similarly verify that many properties of classical set operations also hold true for fuzzy sets, including idempotency: $A \cup A = A$ and $A \cap A = A$, commutativity: $A \cup B = B \cup A$ and $A \cap B = B \cap A$, associativity: $(A \cup B) \cup C = A \cup (B \cup C)$ and $(A \cap B) \cap C = A \cap (B \cap C)$, double negation: $(A^c)^c = A$, and De Morgan's Laws: $(A \cup B)^c = A^c \cap B^c$ and $(A \cap B)^c = A^c \cup B^c$. However, the law of contradiction, $A \cap A^c = \emptyset$ (where $\emptyset$ is the empty set, having membership function 0 everywhere)



μ_{BLUE}	μ_{BLUE^C}
1	0
0.8	0.2
0.6	0.4
0.4	0.6
0.2	0.8
0	1

μ_{GREEN}	μ_{BLUE}	$\mu_{BLUE \cup GREEN}$
1	1	1
1	0.5	1
1	0	1
0.8	0.5	0.8
0.5	0.8	0.8
0.2	0.4	0.4
0	1	1

μ_{GREEN}	μ_{BLUE}	$\mu_{BLUE \cap GREEN}$
1	1	1
1	0.5	0.5
1	0	0
0.8	0.5	0.5
0.5	0.8	0.5
0.2	0.4	0.2
0	1	0

Fig. 6.4 An illustration of the fuzzy set operations complement, union and intersection using color concepts. The tables to the right provide numerical examples for several sample membership function values. Note that for the binary (0 or 1) cases, the fuzzy set operations produce the same results as those used in classical logic.

does *not* hold for fuzzy sets since an element may have nonzero membership in both a fuzzy set and its complement.

The logical operations of complement (not), intersection (and), and union (or) have been defined above by a mathematical operation on the sets' membership functions, as "one minus", "min", and "max", respectively. This is intriguing, since it suggests that other mathematical operations on membership functions may also be interpreted as logical operations on fuzzy sets. Indeed, the arithmetic mean of two membership functions, $(\mu_A + \mu_B)/2$, or geometric mean, $(\mu_A \cdot \mu_B)^{1/2}$ are both frequently used in fuzzy logic algorithms, offering "softened" versions of "or" and "and" as illustrated in Fig. 6.6. Using different exponents and weights can allow further manipulation of

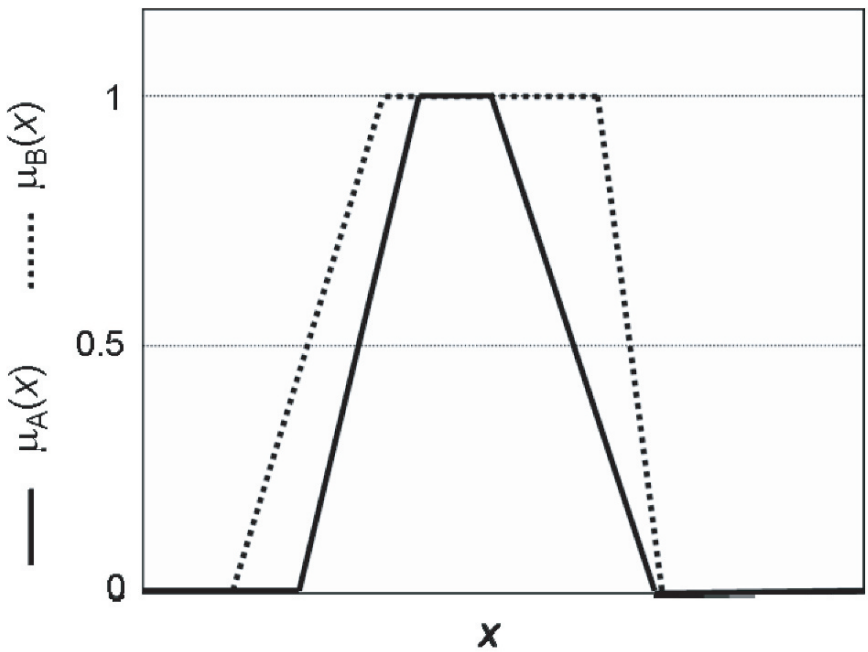


Fig. 6.5 Illustration of fuzzy sets A and B defined over a common domain with $A \subset B$.

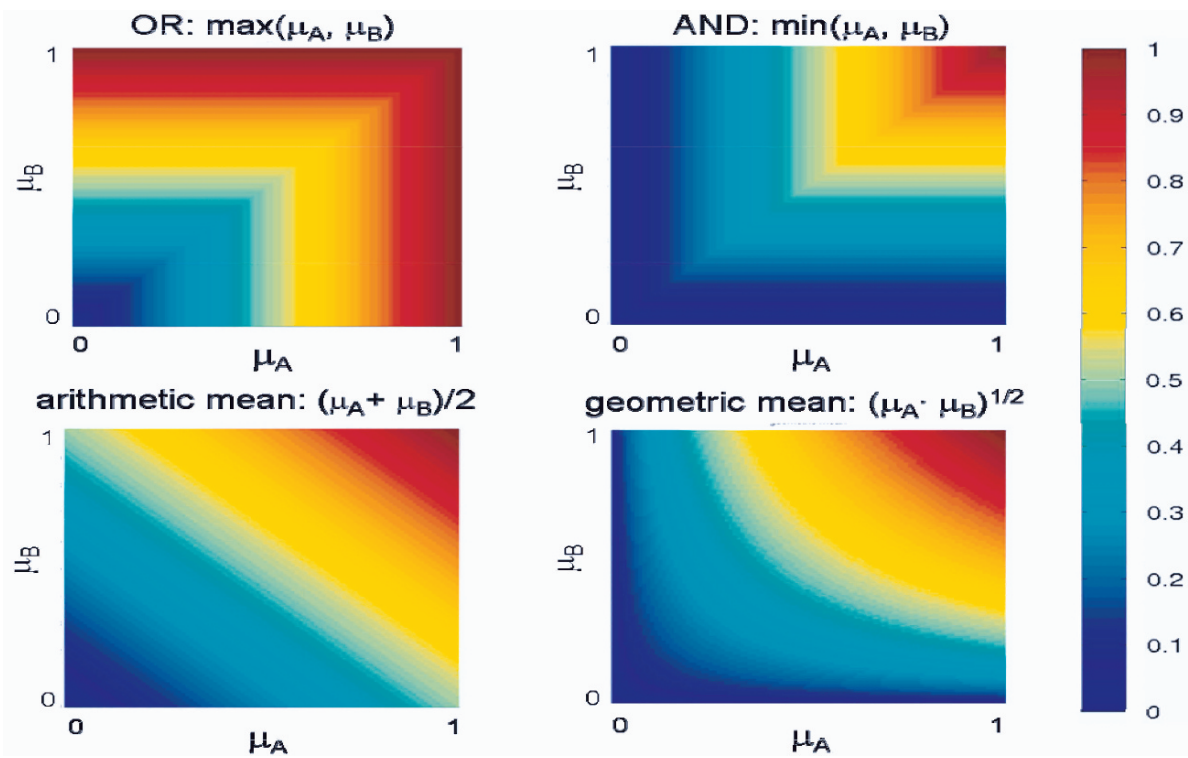


Fig. 6.6 Some functions of two variables that can be used as fuzzy-logical operations. The functions' output values are depicted by colors ranging from 0 (blue) to 1 (red) as illustrated by the colorbar at right.

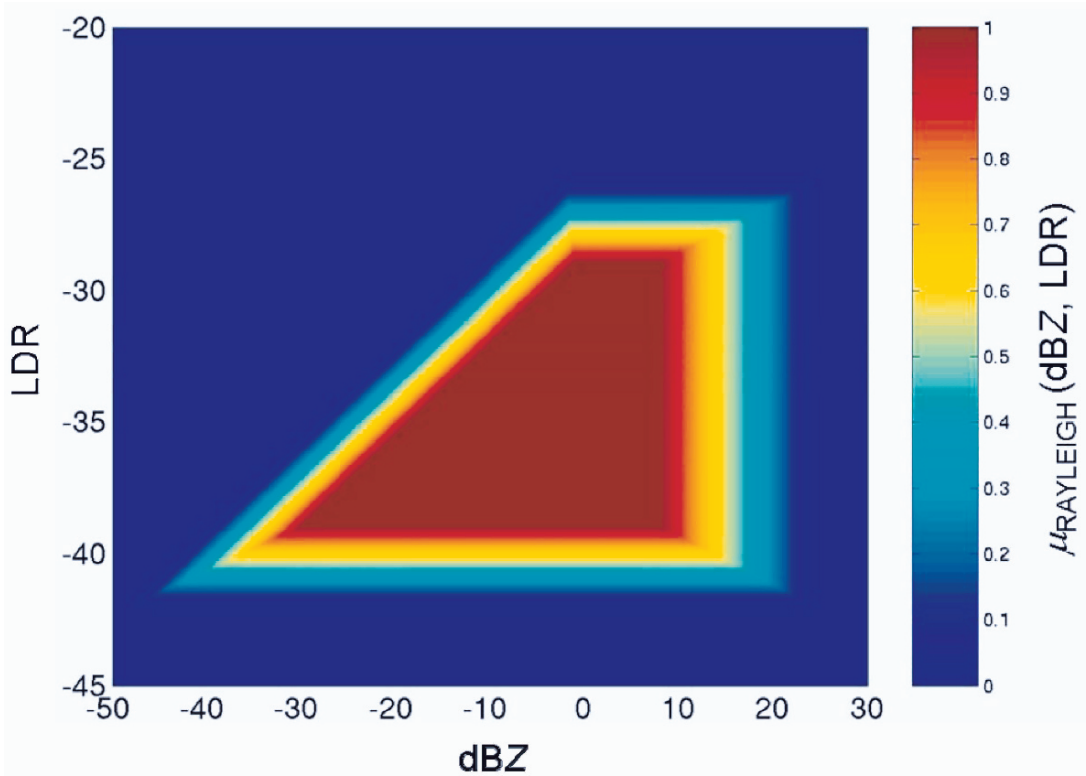


Fig. 6.7 An example 2-D membership function for "W-band radar Rayleigh scattering" as a function of reflectivity and linear depolarization ratio (LDR).

these means' behavior. Many other manipulations of fuzzy sets are possible, leading to great flexibility in the design of fuzzy logic algorithms. However, it is worth reiterating that for fuzzy sets having only binary (0 or 1) membership values, fuzzy logic's standard logical operations are completely equivalent to those for classical sets. Thus, fuzzy logic is equivalent to classical logic for "crisp" fuzzy sets, and fuzzy set theory is truly an *extension* of classical set theory.

Finally, it should be noted that while it is convenient to illustrate fuzzy membership functions as being functions of a single variable, they may actually be functions of arbitrarily many. In environmental science applications, the variables might be multiple attributes or measurements of the environmental state. As an example, consider the problem of determining whether the droplets in a cloud are small enough that they produce Rayleigh scattering of a radar signal, which is important for some remote sensing applications. A possible membership function for "W-band Rayleigh scattering" (W-band is a radar frequency near 95 GHz), based on the radar-measured linear depolarization ratio (LDR) and reflectivity (dBZ), is shown in Fig. 6.7. This fuzzy set is most conveniently defined in 2-D because of the complicated interaction of these two quantities (Vivekanandan et al. 1999).

6.4 Fuzzy Numbers

Although we usually think of numbers as being inherently precise, they often actually represent approximations or are subject to uncertainty. For instance, someone might say "I'll be ready at 7 pm" or "it will take 15 minutes to get there", but the first sentence is often understood to be approximate and the second is an estimate whose accuracy may vary considerably based on travel conditions. One way to propagate this sort of "fuzziness" through a calculation or take it into

account when comparing two quantities is through the concept of *fuzzy numbers*.

Before defining fuzzy numbers, it is helpful to establish some descriptors for different kinds of fuzzy sets. We say that a fuzzy set A is *convex* if its membership function contains no local "dips", or, formally, if for any elements x_1 and x_2 and any number λ between 0 and 1, $\mu_A((1-\lambda)x_1 + \lambda x_2) \geq (1-\lambda)\,\mu_A(x_1) + \lambda\,\mu_A(x_2)$. A fuzzy set A is *normal* if there is at least one element x for which $\mu_A(x) = 1$. Using these concepts, a *fuzzy number* is defined to be a convex, normal fuzzy set with a continuous membership function having the property that $\{x : \mu_A(x) = 1\}$ is either a single element or a connected region (e.g., an interval). Common examples of fuzzy numbers include "triangle" functions and Gaussian bell-curve functions.

Any function f that operates on ordinary numbers may become a mapping of fuzzy numbers through the *extension principle*. If $f : \mathrm{X} \rightarrow \mathrm{Y}$, then the image of a fuzzy set A defined in the space X is a fuzzy set $f(A)$ in the space Y with membership function $\mu_{f(A)}(y) = \sup\{\mu_A(x) | f(x) = y\}$ when y is in the range of f (that is, there is at least one element $x \in \mathrm{X}$ for which $f(x) = y$), or 0 otherwise. Here "sup" stands for *supremum*, the smallest number that is greater than or equal to every member of the set of values within the curly brackets; it is the same as the *maximum* if the set in brackets is "closed." The extension principle basically says that the membership value of $f(A)$ at a point y is the biggest of the membership values $\mu_A(x)$ for the points x that are mapped to y. If f takes multiple arguments, so that $f : \mathrm{X}_1 \times \mathrm{X}_2 \times \ldots \mathrm{X}_n \rightarrow \mathrm{Y}$, then the fuzzy set $A = A_1 \times A_2 \times \ldots A_n$ may be defined by $\mu_A(x_1, x_2, \ldots, x_n) = \min(\mu_{A_1}(x_1), \mu_{A_2}(x_2), \ldots, \mu_{A_n}(x_n))$ and the extension principle says $\mu_{f(A)}(y) = \sup\{\min(\mu_{A_1}(x_1), \mu_{A_2}(x_2), \ldots, \mu_{A_n}(x_n))|$ $f(x_1, x_2, \ldots, x_n) = y\}$ when y is in the range of f, and 0 otherwise.

Computing the image of a fuzzy set can be computationally difficult, particularly for multivariate functions. However, there are some cases in which it is relatively easy. For instance, if f is an invertible function, and y is in the range of f, then $\mu_{f(A)}(y) = \mu_A\left(f^{-1}(y)\right)$. In particular, if f is linear (and nonzero) and A has a piecewise-linear membership function, then $f(A)$ is a piecewise-linear function determined by simply mapping the vertices of the membership function for A via f. For instance, the membership function for the fuzzy number "about 3" shown in Fig. 6.8 has vertices (2.5, 0), (2.875, 1), (3.125, 1), and (4.0, 0); its image under a linear function f is again piecewise linear with vertices (f(2.5), 0), (f(2.875), 1), (f(3.125), 1), and (f(4.0), 0). The image of "about 3" under a monotonic but nonlinear map g may not be piecewise linear, as the example in Fig. 6.8 shows. Nevertheless, a piecewise linear approximation with vertices (g(2.5), 0), (g(2.875), 1), (g(3.125), 1), and (g(4.0), 0) would probably be adequate for many practical applications.

Computing with fuzzy numbers provides a way to propagate uncertainty or ambiguity in input values through a formula or algorithm. For instance, fuzzy numbers may be used to represent environmental quantities subject to small-scale spatial or temporal variability or random measurement noise. If these are then used as inputs to a mathematical model or other formula, the answer will be a fuzzy number whose shape will indicate the spread or uncertainty in the result. For example, a method for remotely detecting the liquid water content L in clouds using measurements from two radars operating at different frequencies is based on a formula that relates L to the range derivative of the difference in measured reflectivity:

$$L(r) \approx \frac{(\mathrm{dB}Z_1(r+\Delta r) - \mathrm{dB}Z_2(r+\Delta r)) - (\mathrm{dB}Z_1(r-\Delta r) - \mathrm{dB}Z_2(r-\Delta r))}{2\,\Delta r\,A_L(r)} \tag{6.1}$$

where r represents a range along the radar beams, Δr is the range spacing between adjacent measurements along a beam, dBZ denotes reflectivity on a decibel scale, and A_L is a differential absorption coefficient whose value is a function of the two radar frequencies and the temperature (Williams and Vivekanandan 2007). The uncertainty in estimating the temperature at r leads to an uncertainty in A_L, and random noise affects each of the dBZ measurements. Using fuzzy numbers to represent each of these quantities yields a fuzzy number $L(r)$ that provides information about the distribution of possible true values

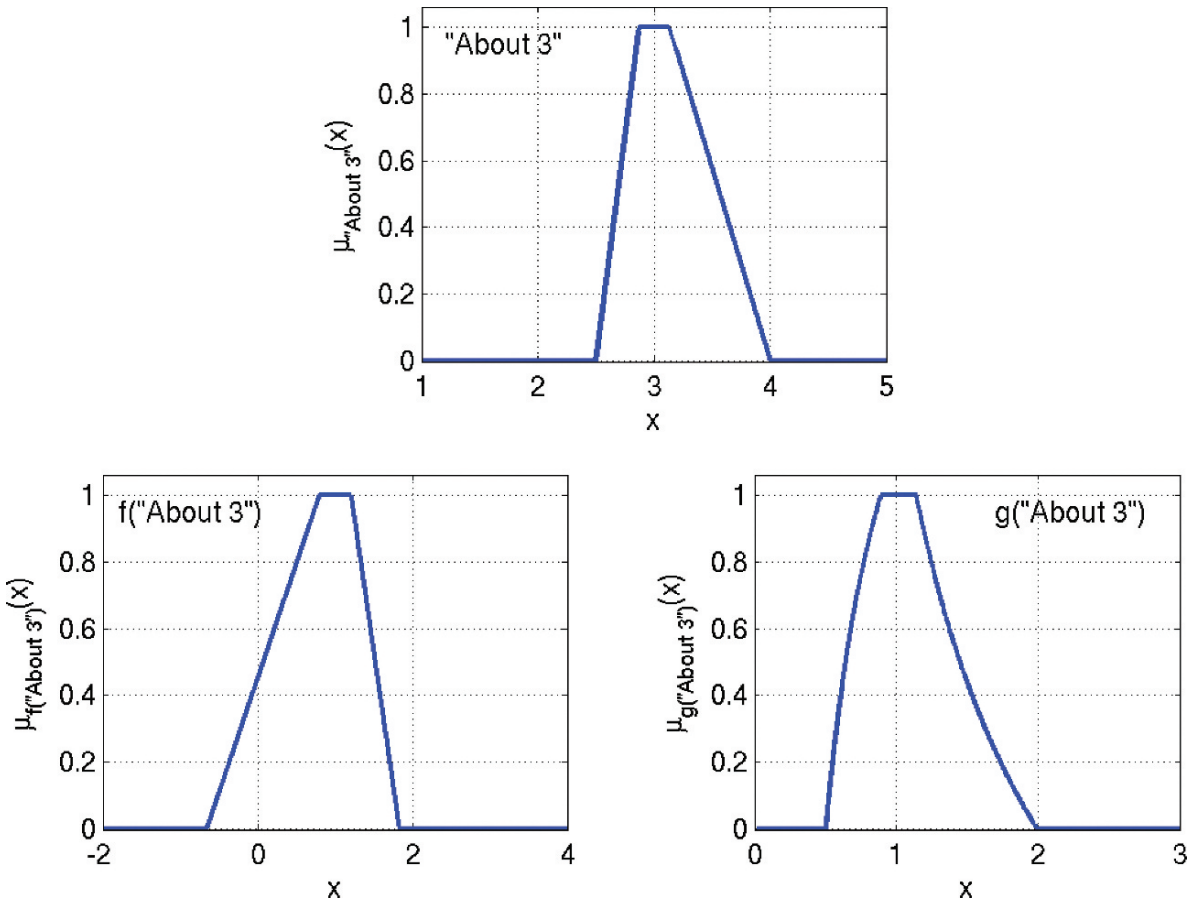


Fig. 6.8 (Top) One possible membership function for a fuzzy number "about 3". (Left) Membership function for f("about 3") where $f(x) = -5/3x + 6$. (Right) Membership function for g("about 3") where $g(x) = 1/(x - 2)$.

of liquid water content. This characterization of uncertainty might be very important to a decision support application – for instance, in assessing the risk that an aircraft flying through the measured region might experience hazardous icing conditions.

6.5 Fuzzy Logic

In classical logic, many logical statements may be represented in terms of set relations. For instance, a common form of reasoning is *modus ponens*, which states that if the sentence "if P, then Q" and the statement "P" both hold true, then it necessarily follows that the statement "Q" must also hold true. The statement "if P, then Q" is equivalent to the set inclusion $A \subset B$, where A is the set of elements for which P is true and B is the set of elements for which Q is true. Said another way, suppose x is an element, e.g., an environmental state, with attributes $a(x)$ and $b(x)$. Then the statement, "if $a(x) \in A$ then $b(x) \in B$" means that the set of elements x for which $a(x) \in A$ is a subset of the elements x for which $b(x) \in B$, that is, $A \subset B$. In terms of characteristic functions, if $\chi_A \leq \chi_B$ and $\chi_A(x) = 1$, then it follows from this statement that necessarily $\chi_B(x) = 1$; on the other hand, if $\chi_A(x) = 0$, nothing is known about $\chi_B(x)$. In fact, it is possible to write an equation for modus ponens in terms of characteristic functions that expresses the value of $\chi_B(x)$ given a value for $a(x)$ and the statement "if $a(x) \in A$ then $b(x) \in B$" ("$A \rightarrow B$" for short). One such equation is $\chi_{a(x),A\rightarrow B}(b(x)) = \min(1, 1 - \chi_A(x) + \chi_B(x))$. To see why, note that if

$a(x) \in A$, then $\chi_A(x) = 1$ and it follows that $\chi_{a(x),A\to B}(b(x)) = \min(1, \chi_B(x)) = \chi_B(x)$, that is, $b(x) \in B$. Otherwise, $\chi_A(x) = 0$ and it follows that $\chi_{a(x),A\to B}(b(x)) = \min(1, 1 + \chi_B(x)) = 1$, representing the convention in classical logic that a logical statement is always held to be true when the premise is not satisfied, or, equivalently, that the statement gives no information about whether $b(x) \in B$ in this case. Thus, the modus ponens equation shows that for each value $a(x)$ in the domain of A, the classical logic statement "$A \to B$" produces a *set* over $b(x)$ in the domain of B; this set may either be B itself or the entire domain of B. Since fuzzy logic is an extension of classical logic, the fuzzy logic equivalent to modus ponens must be defined to also generate such a set.

In fuzzy logic, the statements P and Q are not necessarily either true or false, but may be only partially true. If A and B are fuzzy sets with membership function μ_A representing the degree to which P is true and μ_B representing the degree to which Q is true, then by analogy with classical logic, the statement "if P, then Q" should mean $\mu_A \le \mu_B$. However, this tempting interpretation turns out to be nonsensical. For instance, it might happen that the fuzzy set B is not normal, that is, Q is a statement that is never fully true. Then μ_B is always less than one, but it should still be possible to form a meaningful fuzzy logical statement "$A \to B$" where A is normal, that is, μ_A may be 1 and the premise completely true! For instance, consider the statement "if a dessert includes chocolate, then it is delicious", so that $P =$ "a dessert includes chocolate" and $Q =$ "a dessert is delicious". The fuzzy set A of desserts including chocolate certainly has members with truth value one. On the other hand, the fuzzy set B of "delicious desserts" may not have any incontrovertible members – particularly if a cantankerous professional food critic is setting the scale. The crux of the issue is that fuzzy logic gives information about the consequent Q even when the premise P is not completely true or the consequent itself is ambiguous.

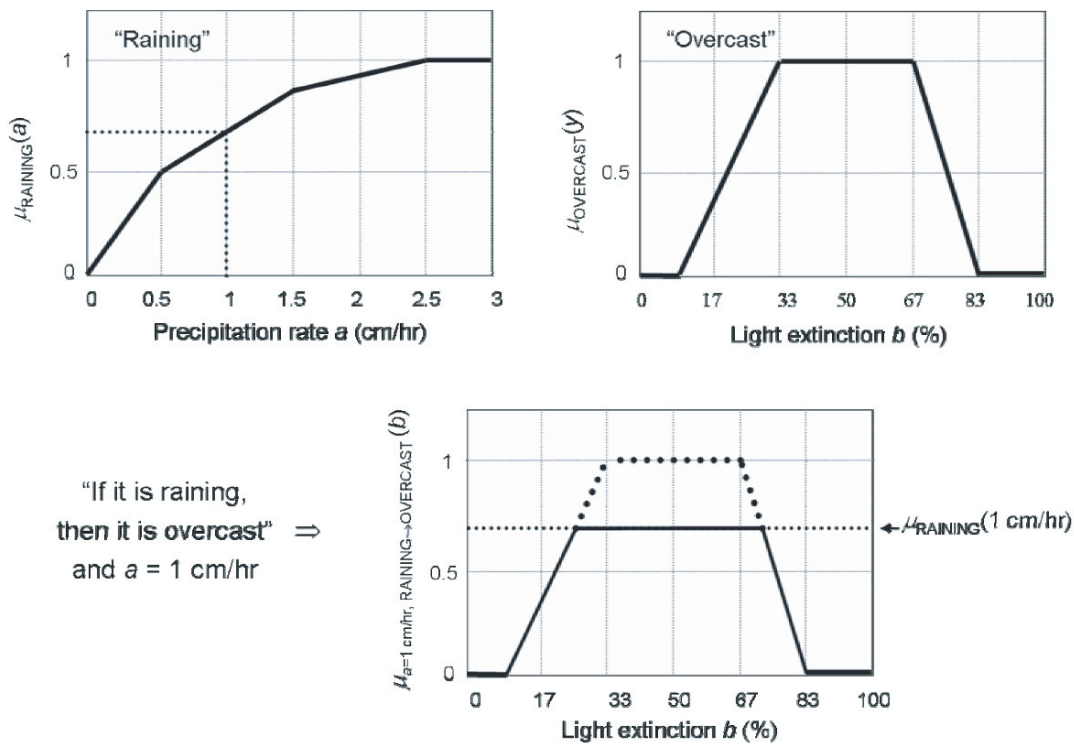


Fig. 6.9 Illustration of Mamdami's fuzzy implication. The top two plots provide possible definitions of the concepts "raining" in terms of precipitation rate and "overcast" in terms of solar light extinction, respectively; the lower plot illustrates the fuzzy set resulting from the statement "If it is raining, then it is overcast" given a measured precipitation rate value of 1 cm/h.

There are a number of different definitions for fuzzy modus ponens that define the resultant fuzzy set $\mu_{a(x),A\to B}(b(x))$ in a way that achieves this. One was proposed by Mamdami (1974): $\mu_{a(x),A\to B}(b(x)) = \min(\mu_A(x), \mu_B(x))$. For a given value of $a(x)$, $\mu_A(x)$ has a specific value between 0 and 1 and the right hand side of this equation represents a fuzzy set over attribute b of x that is equal to $\mu_B(x)$ but truncated at an upper bound of $\mu_A(x)$. In words, this definition of fuzzy logical implication means something like, "if attribute a of x is a member of set A to the extent $\mu_A(x)$ and $A \to B$, then attribute b of x is a member of set B to at most the extent $\mu_A(x)$." Mamdami-style inference is illustrated in Fig. 6.9 for the statement, "If it is raining, then it is overcast." Other definitions for $\mu_{a(x),A\to B}(b(x))$ include one proposed by Zadeh: $\min(1, 1 - \mu_A(x) + \mu_B(x))$; the product: $\mu_A(x)\mu_B(x)$; the bounded product: $\max(0, \mu_A(x) + \mu_B(x) - 1)$; and the Boolean logic implication: $\max(1 - \mu_A(x), \mu_B(x))$.

In most cases, additional pieces of evidence and associated logical statements are necessary to refine the fuzzy set resulting from fuzzy modus ponens. The process of combining different sources of evidence using logical rules and determining a final "crisp" output is called *fuzzy inference*.

6.6 Mamdami-Style Fuzzy Inference

The ultimate purpose of an artificial intelligence algorithm is to map a set of inputs to one or more outputs. In environmental science applications, the inputs often consist of environmental measurements or measurement-derived quantities, and the outputs typically represent an estimate of some attribute of the system's state or a recommended action to be taken. Such mappings may be complex, and will often be nonlinear. Fuzzy inference provides a way to build up a mapping of this sort using logical rules like those often employed in human natural language, as described in the previous section. Because an "if…then" statement involving fuzzy sets applies for any degree of truth of the antecedent and provides only weak information about the consequent, many such fuzzy rules may need to be combined to provide conclusive evidence. The ability to make use of multiple sources of ambiguous information is one of the great strengths of fuzzy logic inference. Not only does it allow information to be used efficiently, but the fact that many different rules are employed means that if some input data are missing, the fuzzy inference system can still function using the remaining rules. For this reason, fuzzy logic algorithms are generally quite robust.

To explore the structure of a fuzzy inference system more concretely, suppose that the goal of the system is to predict a value for a state parameter $u(x)$ based on a number of environmental attributes $a(x), b(x), \ldots, d(x)$. If a number of rules exist connecting these environmental attributes (measurements or derived quantities, for instance) to the state parameter being predicted, then the antecedents and consequents of these rules must be *fuzzified*, i.e., defined as appropriate fuzzy sets over the domain of the relevant attribute. The rules can then be expressed as a set of fuzzy logic statements like those discussed in the previous section, e.g., $A \to R, B \to S, \ldots D \to T$. Using the known values of the various environmental attributes in conjunction with these statements and an appropriate definition of fuzzy implication, each rule will yield a fuzzy set over the domain of the parameter $u(x)$. These resulting fuzzy sets may then be *aggregated* to form a final resultant fuzzy set over $u(x)$. The most common way to perform this aggregation is by taking the *maximum* of the membership functions resulting from the various rules. In this case. the membership function resulting from the input values and fuzzy rules is given by $\mu_{a(x),b(x),\ldots,d(x),A\to R,B\to S,\ldots,D\to T}(u(x)) = \max(\mu_{a(x),A\to R}(x), \mu_{b(x),B\to S}(x), \ldots, \mu_{d(x),D\to T}(x))$. If the Mamdami definition of fuzzy implication is used, for example, this becomes $\mu_{a(x),b(x),\ldots,d(x),A\to R,B\to S,\ldots,D\to T}(u(x)) = \max(\min(\mu_A(x), \mu_R(x)), \min(\mu_B(x), \mu_S(x)), \ldots, \min(\mu_D(x), \mu_T(x)))$ and is called Mamdami-style inference. Other methods for aggregation include taking the sum of the membership functions resulting from the various rules and either capping it at 1 or renormalizing the sum so that the maximum value is 1.

Once the resultant fuzzy set is obtained by aggregating the results of the fuzzy rules, it is often desirable to produce a final, "crisp" estimate of the value $u(x)$ through a process called *defuzzification*. One way to perform defuzzification is to compute the centroid of

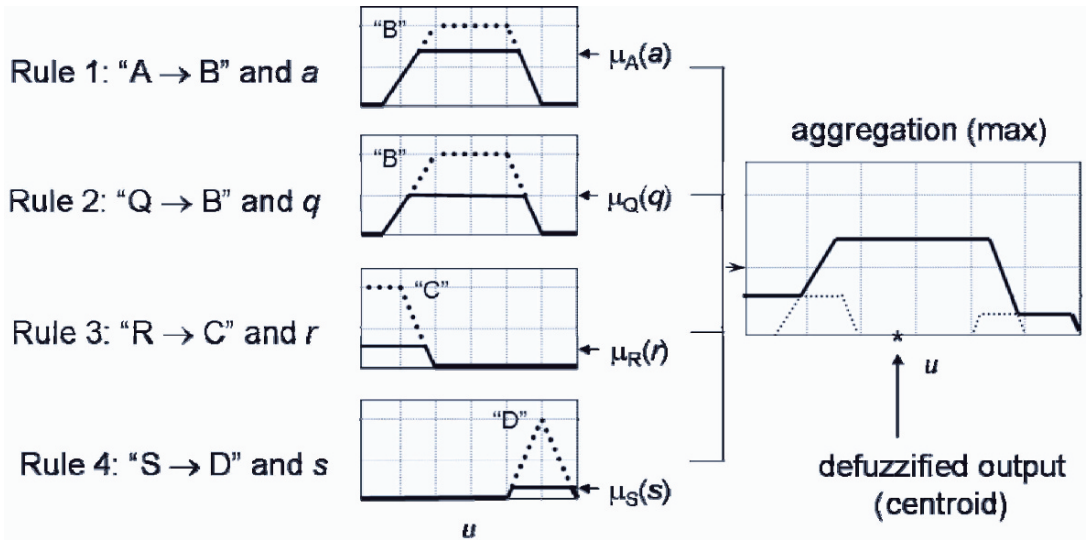


Fig. 6.10 Illustration of Mamdami-style inference for determining a quantity u given four fuzzy rules and input values a, q, r, and s. Aggregation is performed by taking the maximum of the four truncated sets and defuzzification is achieved by computing the centroid of the aggregated fuzzy set.

the aggregated fuzzy set:

$$\int u(x)\, \mu_{a(x),b(x),\ldots,d(x),A\to R,B\to S,\ldots,D\to T}\,(u(x))\, dx \Big/ \int \mu_{a(x),b(x),\ldots,d(x),A\to R,B\to S,\ldots,D\to T}\,(u(x))\, dx \quad (6.2)$$

This is the method illustrated in Fig. 6.10. Other methods for defuzzification include taking the value of $u(x)$ for which $\mu_{a(x),b(x),\ldots,d(x),A\to R,B\to S,\ldots,D\to T}(u(x))$ obtains its maximum, or computing the centroid over only those values for which it exceeds some threshold.

A concrete example may help clarify the Mamdami fuzzy inference procedure. Suppose that a certain teenager, who is in charge of watering the lawn each Saturday, appears to need some assistance in knowing how much to water each week. If you had a soil moisture sensor, you could devise a formula to prescribe a precise watering time. But without a sensor, you might come up with the following simple rules as a guide:

Rule 1: If the weather has been cool or wet, water a little.
Rule 2: If the weather has been hot or dry, water a lot.

These are precisely the kind of natural language rules that can be turned into a fuzzy logic expert system. The first step is *fuzzification*, that is, defining the appropriate fuzzy sets for the concepts involved. *Cool* and *hot* are weather attributes that might naturally be defined based on the average temperature measured over the past week, t; see Fig. 6.11. Similarly, *dry* and *wet* are precipitation concepts that could be defined in terms of the total precipitation over the past week, p. Finally, watering *a little* or *a lot* in this context could be defined by the number of hours the water is to be left on, h. Now on any given Saturday, a review of weather records over the past week will yield t and p. Recalling that the fuzzy set union ("or") may be represented by the maximum of the sets' membership functions, it follows that $\mu_{cool\ or\ wet}(\text{week}) = \max(\mu_{cool}(t), \mu_{wet}(p))$ and $\mu_{hot\ or\ dry}(\text{week}) = \max(\mu_{hot}(t), \mu_{dry}(p))$. To evaluate Rule 1 under Mamdami-style fuzzy inference, we compute the antecedent, $\mu_{cool\ or\ wet}(\text{week})$, and use it to "cap" the membership function of the consequent, $\mu_{a\ little}(h)$, resulting in the fuzzy set $\mu_1(h) = \max(\min(\mu_{cool\ or\ wet}(week), \mu_{a\ little}(h)))$. Similarly, evaluating Rule 2 yields the fuzzy set $\mu_2(h) = \max(\min(\mu_{hot\ or\ dry}(week), \mu_{a\ lot}(h)))$. Next we must *aggregate* the results of these two rules, for instance, by taking their maximum: $\max(\mu_1(h), \mu_2(h))$. Finally, we wish to determine a precise number of hours to water, which we do through *defuzzification*. This may be accomplished, for instance, by taking the centroid $\int h\, \max(\mu_1(h),\ \mu_2(h))\, dh \,/\, \int \max(\mu_1(h), \mu_2(h))\, dh$,

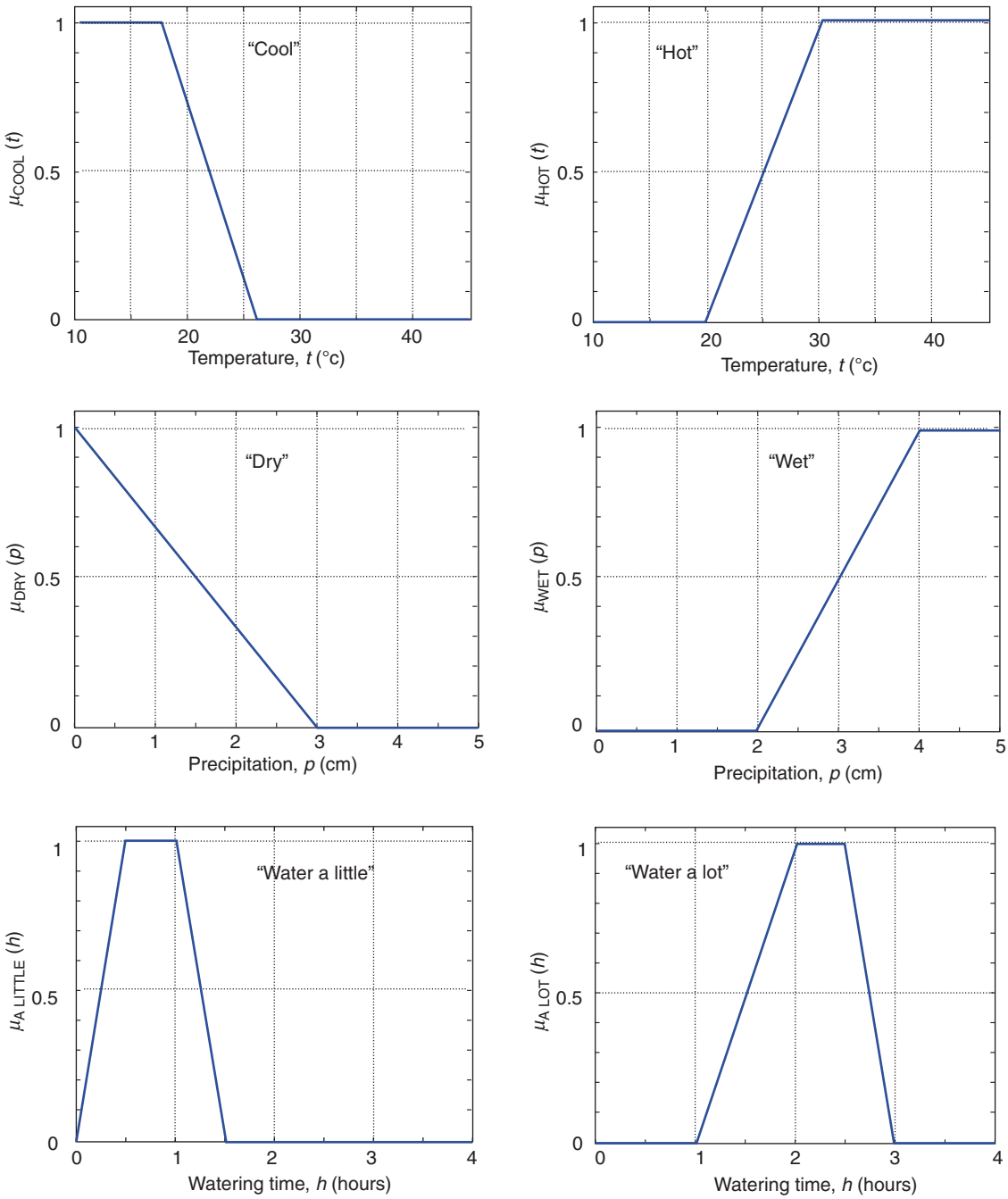


Fig. 6.11 Fuzzy sets for use in determining watering time: temperature concepts (top), precipitation concepts (middle), and watering time concepts (bottom).

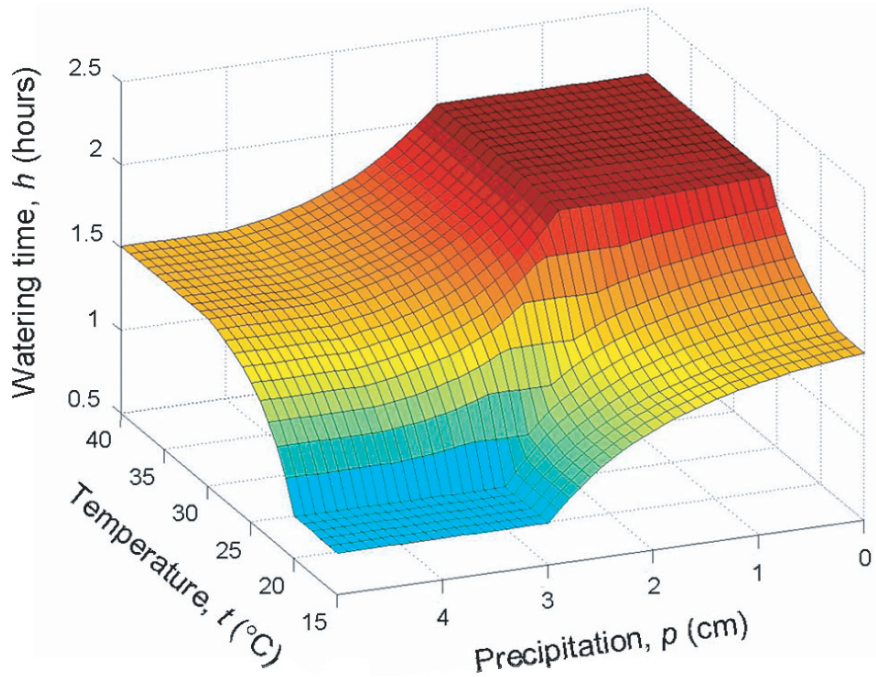


Fig. 6.12 Watering time as a function of the previous week's temperature and precipitation, produced by substituting pairs of values of t and p into the example Mamdami-style fuzzy logic inference system described in the text.

where the integrals are computed via an appropriate sum or quadrature. The ultimate result of this inference procedure is the *function* depicted in Fig. 6.12 that maps a week's average temperature t and total precipitation p into a prescribed number of hours h to water the lawn. The function is surprisingly complex given the simplicity of the rules, and is nonlinear despite the fact that all the membership functions are piecewise-linear. The addition of other rules could help refine it further. For instance, specifying that no watering is needed if it is cool and wet would allow the watering time to approach zero in those conditions, which the initial function does not. And including linguistic rules involving other factors, such as soil type, season of year, or type of grass used in the lawn, could make the function more general. Nevertheless, the fact that Mamdami-style fuzzy inference based on two simple rules produces a function having basically the correct behavior clearly illustrates the power of this approach.

6.7 Takagi-Sugeno Fuzzy Inference

The language of science is of course not only linguistic, but also mathematical. Thus, another form of human scientific reasoning involves determining when the environmental conditions are those in which a known physical relationship applies. For example, photosynthesis in plants may be limited by light, water, carbon dioxide or nutrient availability, so different models for photosynthesis may be applicable under different soil, weather and atmospheric conditions. One might apply each of these models to make a prediction of photosynthesis, then perform a weighted average of the predictions using weights that represent their degree of applicability based on the current environmental conditions. A version of fuzzy inference that accommodates this form of reasoning is called Takagi-Sugeno-style fuzzy inference, named for the researchers who proposed it (Takagi and Sugeno 1985).

In Takagi-Sugeno inference, a linguistic rule is used to determine the degree of applicability of a given formula for determining the variable of interest. The consequent of a Takagi-Sugeno fuzzy rule is not a fuzzy set as it is for Mamdami-style inference, but instead a direct estimate of the variable determined by the formula. So if the goal of the fuzzy inference system is to predict a value for a state parameter $u(x)$ based on a number of environmental attributes $a(x)$, $b(x), \ldots,$ $d(x)$, a fuzzy rule might then have the structure, "If A, then $u(x) = f(a(x), b(x), \ldots, d(x))$", where A is

a fuzzy set dependent on some number of environmental attributes. (In a common formulation of Takagi-Sugeno fuzzy inference, the function f is assumed to be linear and to use only the variables involved in the linguistic rule, but these restrictions are not necessary.) Given n rules of the form, "If A_i, then $u(x) = f_i(a(x), b(x), \ldots, d(x))$", the resulting estimate of $u(x)$, denoted $\hat{u}(x)$, is then determined by the weighted average

$$\hat{u}(x) = \frac{\sum_{i=1}^{n} \mu_{A_i}(x)\, f_i(a(x), b(x), \ldots, d(x))}{\sum_{i=1}^{n} \mu_{A_i}(x)} \tag{6.3}$$

In some cases, there may be some rules which have greater relevance than others even when their antecedents are equally true. In this case, a researcher might assign an importance weight w_i to each rule and compute the estimate of $u(x)$ via the formula

$$\hat{u}(x) = \frac{\sum_{i=1}^{n} w_i\, \mu_{A_i}(x)\, f_i(a(x), b(x), \ldots, d(x))}{\sum_{i=1}^{n} w_i\, \mu_{A_i}(x)} \tag{6.4}$$

Note that no additional defuzzification step is needed for Takago-Sugeno fuzzy inference because the result is the numerical estimate $\hat{u}$, not a fuzzy set. This makes Takago-Sugeno fuzzy inference comparatively efficient, while the ability to use arbitrary functions f as the consequents of the fuzzy rules makes it quite flexible. In addition, algorithms using this architecture are inherently robust because if some data are missing, the remaining data may still be sufficient to compute (6.4) with some values of i left out.

A special form of Takagi-Sugeno fuzzy inference occurs when the functions f_i are simply constant values, that is, when the fuzzy rules have the form, "If A, then $u(x) = c$." In this case (6.3) may be interpreted as averaging a number of numerical predictions, with the weight of each prediction being given by the degree of truth of the antecedent to each rule. It is interesting to note that the final result in this case is the same as it would be if the consequents are interpreted as fuzzy sets and Mamdami-style fuzzy inference is employed along with the centroid method of defuzzification. For example, Takagi-Sugeno style rules for the watering time example from the previous section might be written as:

Rule 1: If the weather has been cool or wet, watering time $h_1 = 0.75$ h.
Rule 2: If the weather has been hot or dry, watering time $h_2 = 2.15$ h.

Then the resulting estimate of required watering time is given by the function:

$$\begin{aligned}\hat{h}(t, p) &= \frac{\mu_{cool\ or\ wet}(t, p)\,(0.75) + \mu_{hot\ or\ dry}(t, p)\,(2.15)}{\mu_{cool\ or\ wet}(t, p) + \mu_{hot\ or\ dry}(t, p)} \\ &= \frac{\max(\mu_{cool}(t), \mu_{wet}(p))(0.75) + \max(\mu_{hot}(t), \mu_{dry}(p))(2.15)}{\max(\mu_{cool}(t), \mu_{wet}(p)) + \max(\mu_{hot}(t), \mu_{dry}(p))}\end{aligned} \tag{6.5}$$

where the membership functions for *cool*, *wet*, *hot*, and *dry* are again defined by the plots in Fig. 6.11. This function $\hat{h}$ is nearly indistinguishable from the function resulting from the Mamdami-style inference system described in the previous section and plotted in Fig. 6.12. Indeed, the Mamdami-style system would be identical to (6.5) if the fuzzy sets for "water a little and "water a lot" were replaced by their "typical" values, that is, the singleton sets $\mu_{a\ little}(h) = \begin{cases} 1 \text{ if } h = 0.75 \\ 0 \text{ otherwise} \end{cases}$ and $\mu_{a\ lot}(h) = \begin{cases} 1 \text{ if } h = 2.15 \\ 0 \text{ otherwise} \end{cases}$, turning the centroid defuzzification step into the simple weighted mean. A Mamdami-style inference system can frequently be simplified using this technique without substantial loss of information, particularly if the consequent fuzzy sets are symmetric or nearly so and do not overlap substantially.

6.8 Fuzzy Consensus Methods

A somewhat simplified form of Takagi-Sugeno fuzzy inference may be interpreted as one of the class of artificial intelligence techniques called *consensus methods* or *mixtures of experts*. In this interpretation, each member of a group of "experts" evaluates various

attributes of the environmental state and makes a prediction of some variable of interest. The results are then averaged, using weights for each expert's prediction based on some measure of his or her reputation or past skill. In addition, the experts might accompany each prediction with a 0 to 1 measure of their confidence in it, with greater confidence being expressed when the required data are available and of good quality and the theory or experience being applied by the expert is judged appropriate to the conditions at hand. If the reasoning of each expert from evidence to prediction is described by a function f, this procedure corresponds precisely to that described in (6.4), but with the fuzzy set membership values $\mu_{A_i}(x)$ replaced by each expert's "confidence", c_i:

$$\hat{u}(x) = \frac{\sum_{i=1}^{n} w_i\, c_i(x) f_i(x)}{\sum_{i=1}^{n} w_i\, c_i(x)} \tag{6.6}$$

Said another way, the fuzzy consensus reasoning formula (6.6) is a special version of Takagi-Sugeno fuzzy inference in which the antecedent A_i for each rule is taken to be the fuzzy set, "the required data are of good quality and the predictive function is appropriate for this scenario." Many applications in the environmental sciences are naturally handled using fuzzy consensus reasoning, taking as inputs several environmental measurements or features, using an assortment of methods to predict a variable of interest, and then combining the results. For instance, in forecasting thunderstorm initiation, researchers might utilize data from satellite-measured cloud-top growth rates, numerical weather models, and radar measurements, each of which can be used to give a prediction of whether initiation is likely to occur, and then combine them to obtain a more reliable forecast than any of the inputs could provide by themselves.

In many applications, assigning the confidences c_i is a very important part of the fuzzy consensus algorithm. In fact, the confidences are often computed using data quality values and assessments of relevance that are themselves determined by separate fuzzy logic algorithms. The assignment of a confidence to each prediction also allows the final output given by the fuzzy logic algorithm to itself be assigned a confidence value that will aid users in interpreting it for decision making purposes. For instance, if an atmospheric turbulence detection algorithm estimates severe turbulence in a region but has low confidence in this assessment, it might not make sense to re-route air traffic around the area without other confirming information that a hazard is present. The confidence associated with the estimate in (6.6) might be computed as

$$\hat{c}(x) = \frac{\sum_{i=1}^{n} w_i\, (c_i(x))^{m+1}}{\sum_{i=1}^{n} w_i\, (c_i(x))^{m}} \tag{6.7}$$

if any product $w_i c_i(x) > 0$, and 0 otherwise, where $m \geq 0$. For example, if one chooses $m = 0$, the formula (6.7) simply produces the weighted-mean confidence. If $m > 0$, then the confidences become part of the averaging weights, and as $m \to \infty$, $\hat{c}(x) \to \max_{\{i \mid w_i > 0\}} c_i(x)$. More generally, since confidences are equivalent to fuzzy membership function values, they may also be combined using fuzzy-logical operations, such as AND (min) or a weighted geometric average such as

$$\hat{c}(x) = \prod_{i=1}^{n} c_i(x)^{w_i / \sum_{k=1}^{n} w_k} \tag{6.8}$$

which yields a large final confidence $\hat{c}$ only if all of the input confidences c_i with $w_i > 0$ are not too small.

Returning to the example of determining the optimal lawn watering time based on the previous week's temperature and precipitation, predictive functions or *interest maps* based on t and p alone might be chosen as depicted in Fig. 6.13; although these happen to be piecewise-linear functions, that is for illustrative purposes and is not a requirement. Suppose that for a particular week both predictions have equal confidence (say 1), and that the temperature interest map is given a weight of 0.4 and the precipitation interest map a weight of 0.6. Then the function resulting from applying the fuzzy consensus reasoning formula (6.6) is displayed in Fig. 6.14. Note that Fig. 6.14 is quite similar to Fig. 6.12 except that it is piecewise linear (being a linear combination of piecewise linear functions in this example). If the confidence in t and p were not equal, then the interest map with higher confidence would begin to dominate. Of course, in a practical application there would ideally be many inputs providing somewhat redundant information so that loss of confidence in one or two input variables would only slightly degrade performance.

Two applications of the fuzzy consensus reasoning technique deserve special mention. The first is an

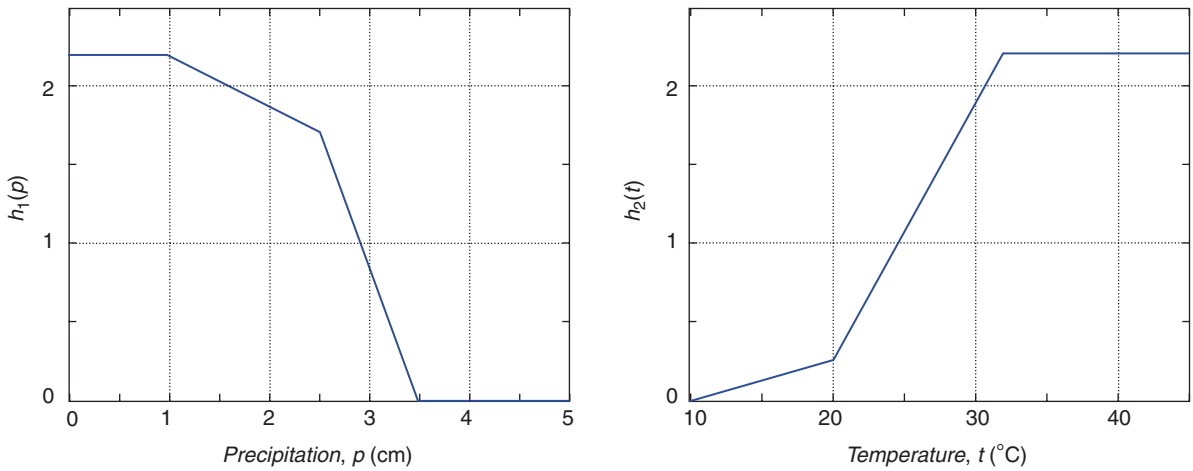


Fig. 6.13 Possible interest maps for determining watering time given precipitation (left) or temperature (right) for use in a fuzzy consensus reasoning system.

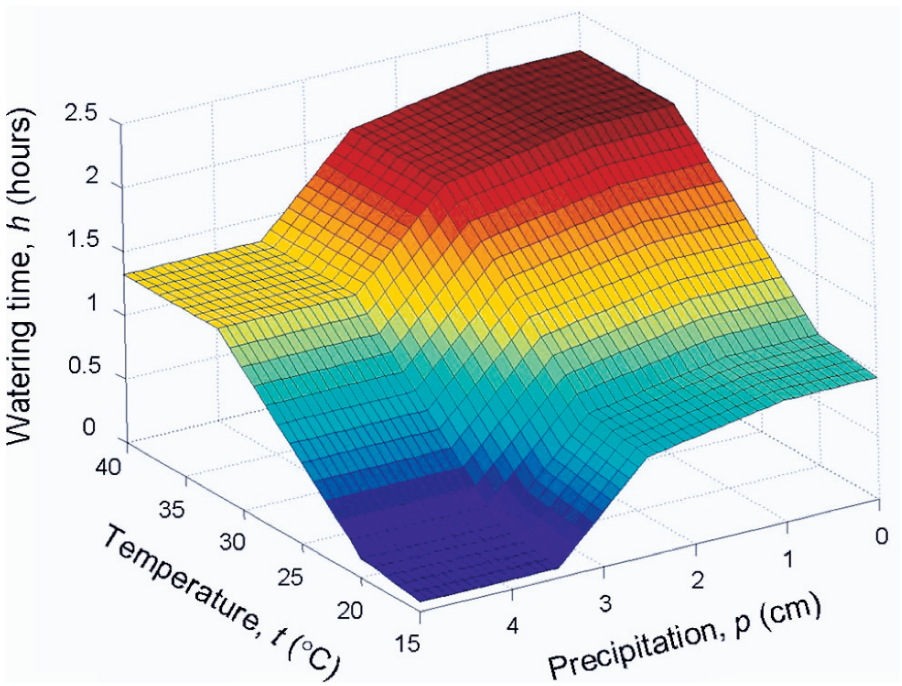


Fig. 6.14 Watering time as a function of t and p from the fuzzy consensus reasoning example described in the text.

application to data smoothing or interpolation. Suppose that a quantity of interest is available over some domain (usually spatial or spatio-temporal), and each value x_i is assigned a confidence estimate c_i, e.g., from a data quality control procedure. A standard smoothing or interpolation approach is to convolve a smoothing kernel, say a Gaussian, with the data field. Fuzzy consensus smoothing additionally makes use of the confidences to reduce the influence of lower-quality data, and also produces a confidence estimate for each smoothed point. For each target location, the smoothing kernel is "centered" at that point and the weight w_i for each data point x_i in the neighborhood is determined by the value of the smoothing kernel there. Then the smoothed value may be computed via an application of (6.6), with $f_i(x)$ replaced with the data value x_i

and $c_i(x)$ replaced with c_i; the confidence associated with the smoothed value may be computed via (6.7). When the quality control procedure is imperfect so that some outliers are assigned nonzero confidence, then the data points in the neighborhood may first be sorted and the top and bottom $p\%$ of the corresponding $w_i c_i$ set to 0 before (6.6) and (6.7) are applied, where p is between 0 and 50%. This method may be referred to as *fuzzy confidence-weighted trimmed-mean kernel smoothing*. If $p = 0$, no trimming occurs, while if $p = 50\%$, the method becomes a confidence-weighted median.

A closely related application is a fuzzy version of the popular k-nearest neighbor (k-NN) algorithm for generalizing from examples. In the classical k-NN, a set of labeled data are used to infer the class of a new example based on the class occurring most frequently among the k nearest labeled points. For instance, the data points might consist of vectors having as components measurements from various sensors (e.g., temperature, pressure, humidity) recorded at 9 am on a number of different days, and the classes could be what sort of weather was observed that afternoon (e.g., clear, cloudy, light rain, thunderstorm). Given a value of k, a distance metric, and a vector of sensor measurements from a subsequent morning, the k-NN algorithm would determine a weather prediction based on the historically best-matching observed class among the k nearest neighbors to that data vector. In the fuzzy version of k-NN, the class labels of the historical points are replaced by fuzzy class membership values between 0 and 1, and the membership of the new example is given by a distance- and membership-weighted consensus of the k nearest neighbors. More precisely, suppose a collection of data vectors $\{\mathbf{x}_j\}$ have fuzzy class memberships μ_{C_i} for classes $\{C_i | 1 \leq i \leq N\}$, k is chosen as the number of neighboring points to consider, and $m > 1$ (smaller values correspond to "tighter", more localized influence). A metric d must be specified so that $d(\mathbf{x}_i, \mathbf{x}_j)$ represents the "distance" between the two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. Common choices for d are the standard Euclidean distance (the square root of the sum of squared differences of the vector elements) or the Manhattan distance (the sum of the absolute values of the vector element differences). It is sometimes useful to first re-scale the data so that the range of values for each vector component is approximately the same. If $\{\mathbf{y}_n | 1 \leq n \leq k\}$ are the k points from the set $\{\mathbf{x}_j\}$ that are closest to a new data vector $\mathbf{x}$, then fuzzy k-NN assigns the class membership of $\mathbf{x}$ for each class C_i via the formula

$$\mu_{C_i}(\mathbf{x}) = \frac{\sum_{n=1}^{k} \mu_{C_i}(\mathbf{y}_n)\, d(\mathbf{x}, \mathbf{y}_n)^{-\frac{2}{m-1}}}{\sum_{n=1}^{k} d(\mathbf{x}, \mathbf{y}_n)^{-\frac{2}{m-1}}} \tag{6.9}$$

If the original class memberships are defined so that their sum for each data point is 1, that is, $\sum_{i=1}^{N} \mu_{C_i}(\mathbf{x}_n) = 1$ for each $\mathbf{x}_n$, then it can be shown that $\sum_{i=1}^{N} \mu_{C_i}(\mathbf{x}) = 1$ as well. In contrast to the standard k-NN algorithm, which yields only the best-matching class, the class memberships $\mu_{C_i}(\mathbf{x})$ returned by fuzzy k-NN provide much richer information about the ambiguity of the similarities identified between the new data and the historical examples. In the weather forecasting example, the historical data class memberships could accommodate ambiguities in the observed weather (e.g., the *degree* of cloudiness), and the memberships assigned via (6.9) would provide some notion of the uncertainty in the nearest neighbor prediction. Moreover, even if the class with the highest membership is ultimately chosen as the final "defuzzified" prediction, the fuzzy k-NN method is somewhat more nuanced than the original version because it weighs the neighborhood data closest to the point being classified more strongly than more distant – and hence potentially less relevant – examples. For instance, in Fig. 6.15, classical k-NN with $k = 5$ assigns the point at "x" to the class "green", since the three of the five nearest points (shown in the dotted circle) are labeled as green. Fuzzy k-NN assigns non-zero memberships to both "blue" and "green", with the precise values depending on the distance weighting function and value of m chosen. If m is small, membership in "blue" could be greatest since a blue point is closest to the "x".

6.9 Fuzzy *c*-Means Clustering

Fuzzy c-means (FCM) clustering is an algorithm for partitioning a multivariate dataset into a prespecified number of fuzzy "clusters", each represented by a fuzzy set defined over the points in the dataset. It

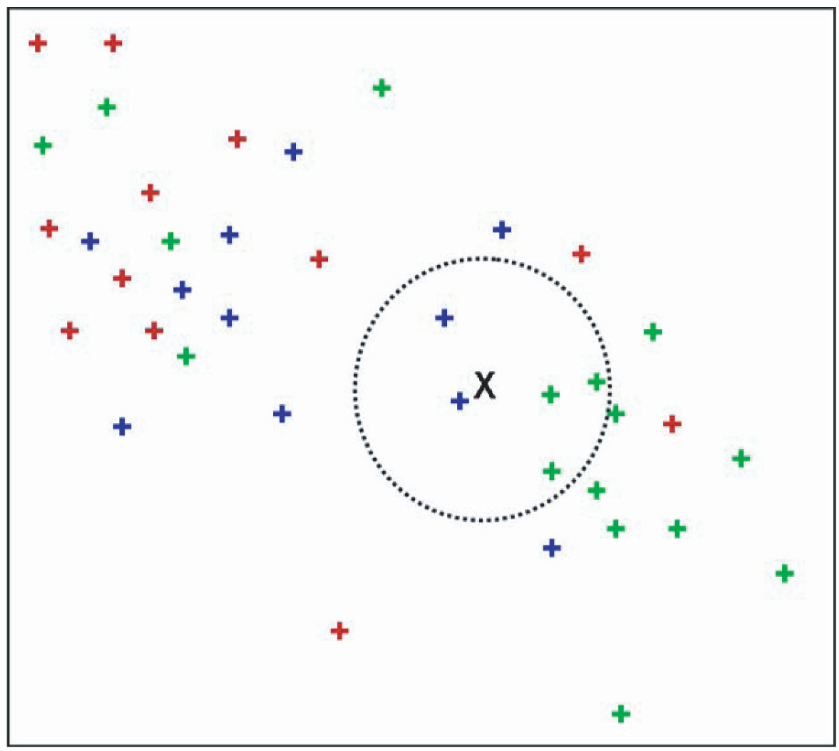


Fig. 6.15 k-nearest neighbor example for three classes (red, green and blue) and $k = 5$.

is a fuzzy variant of the familiar classical method called k-means clustering, which separates data into k "crisp" sets. While the result of k-means clustering is a partition of the data points into disjoint classical sets, under FCM clustering the data points may have partial membership in several different fuzzy sets. Both clustering methods are unsupervised learning methods that find patterns in data without the benefit of "labeling" by a human expert. In environmental science applications, the points being clustered will usually be vectors comprised of several measurements or derived quantities that represent features of the environmental state at a particular time and location. Clustering can reveal the different major domains or attractors in a dynamical system – weather patterns, for example – which may then be analyzed separately to determine their important characteristics. Using fuzzy sets in place of classical ones makes for a more robust clustering algorithm and may provide richer information about the data.

Fuzzy c-means clustering begins with c (a predetermined number) cluster centers, or *prototypes*, which may be arbitrarily initialized or chosen based on prior knowledge of the data. The distance from each of the dataset points to each of the cluster prototypes is computed, and each point is assigned a membership in each of the c clusters based on these distances. New prototypes for each fuzzy cluster are then computed by taking the cluster-membership-weighted centroid of all the points, and the process is repeated until the change in the prototypes becomes small. Figure 6.16 shows an example of fuzzy c-means clustering used to find two clusters in two-dimensional data.

More formally, suppose that the dataset consists of N vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$. As in the k-nearest neighbor technique described earlier, the FCM algorithm requires that a metric d is specified so that $d(\mathbf{x}, \mathbf{y})$ represents the "distance" between vectors $\mathbf{x}$ and $\mathbf{y}$. A number c is selected for the number of fuzzy clusters desired, and initial prototype points $\mathbf{v}_1, \ldots, \mathbf{v}_c$ are specified or chosen randomly. A parameter $m > 1$ is chosen to represent the "tightness" of the clusters: values of m near one will produce more distinct or crisp clusters, while larger values of m will allow more overlap or "fuzziness". Finally, a convergence threshold $\varepsilon > 0$ is chosen to determine when to stop the iteration, which proceeds as follows:

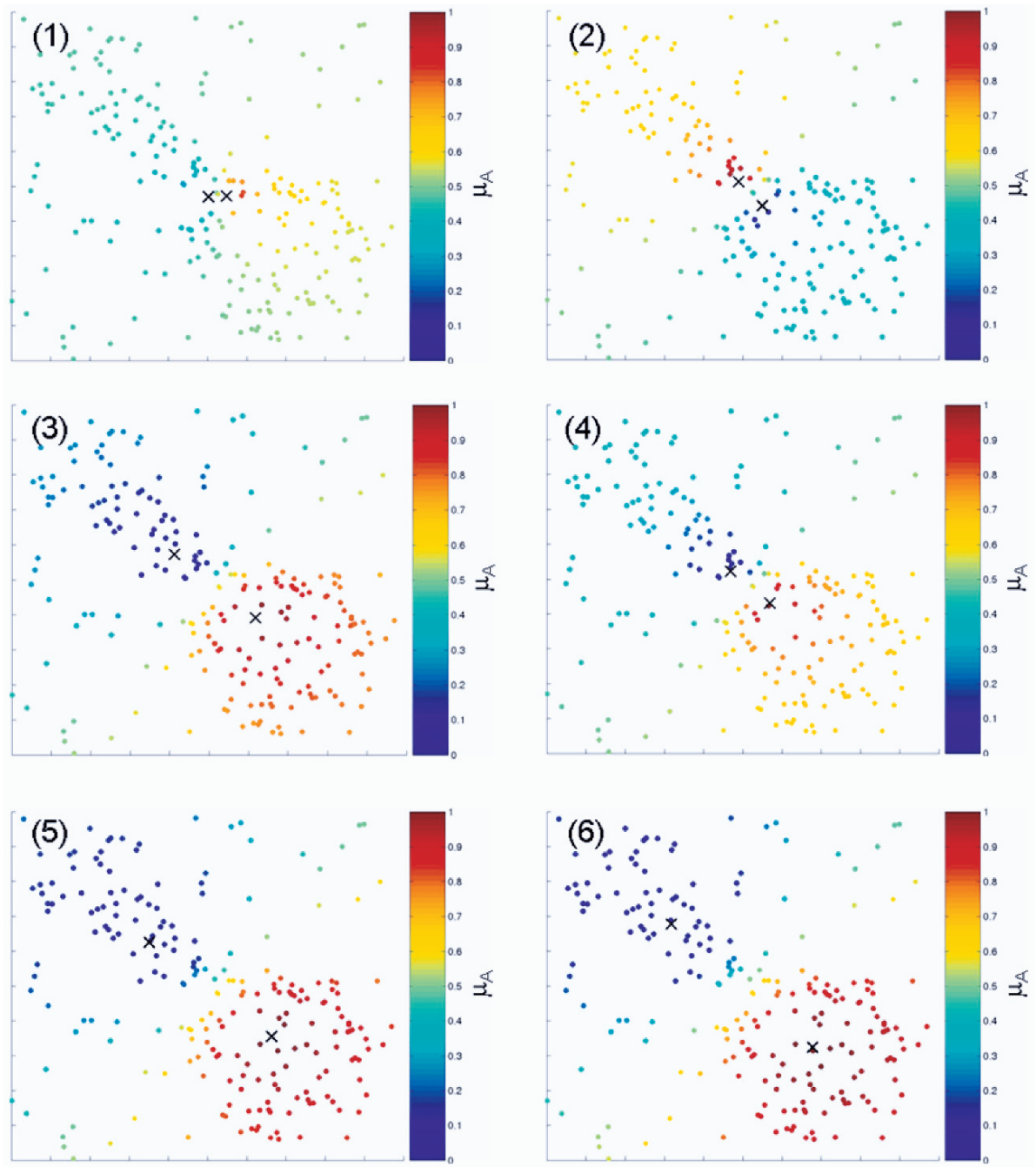


Fig. 6.16 Fuzzy *c*-means clustering example with two fuzzy sets and six iterations shown. The points are colored according to the membership in set "A", so that points with high membership in set "A" are colored red and points with low membership in set "A" (hence high membership in set "B") are colored blue. An "x" marks each cluster center.

1. Define the membership of each data point $\mathbf{x}_k$ in the ith cluster, C_i by

$$\mu_{C_i}(\mathbf{x}_k) = \left(\sum_{j=1}^{c} \left(\frac{d(\mathbf{x}_k, \mathbf{v}_i)}{d(\mathbf{x}_k, \mathbf{v}_j)} \right)^{\frac{1}{m-1}} \right)^{-1}. \quad (6.10)$$

2. Compute new cluster prototypes $\mathbf{v}_i$ via

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N} \mathbf{x}_k \left(\mu_{C_i}(\mathbf{x}_k)\right)^m}{\sum_{k=1}^{N} \left(\mu_{C_i}(\mathbf{x}_k)\right)^m}. \quad (6.11)$$

3. If the absolute value of the change in each cluster prototype is less than ε for every vector element, then the iteration stops. Otherwise, return to step (1).

The final result is a set of fuzzy clusters C_i defined in such a way that for each data vector $\mathbf{x}_k$, $\sum_{i=1}^{c} \mu_{C_i}(\mathbf{x}_k) = 1$, that is, the total membership of $\mathbf{x}_k$ in all c fuzzy clusters is one.

As is true for many machine learning techniques, the choice of the number of clusters, c, and the "fuzziness" parameter m is a bit of an art form, and may require a trial-and-error approach to get meaningful results. And while there are formulas in the literature for determining the "goodness" of a given clustering, its utility may really be dependent on the final application. Note also that the FCM algorithm will not necessarily identify the same fuzzy clusters in different runs unless the initial prototypes are the same each time; thus, a careful choice of the initial prototypes or performing a number of independent runs may be worthwhile.

In addition to identifying structures in data, fuzzy clustering might also provide an important step to developing a fuzzy inference system when human expertise in solving a problem is incomplete. As an example, using clustering to identify different "domains" in weather sensor data might aid in creating a forecast based on those data by training a different predictive model (a multilinear fit, for example, or even a neural network) separately on each cluster. Then a Takagi-Sugeno-style fuzzy inference system could be constructed that combines the various models based on the degree of membership of a point in the antecedent fuzzy cluster.

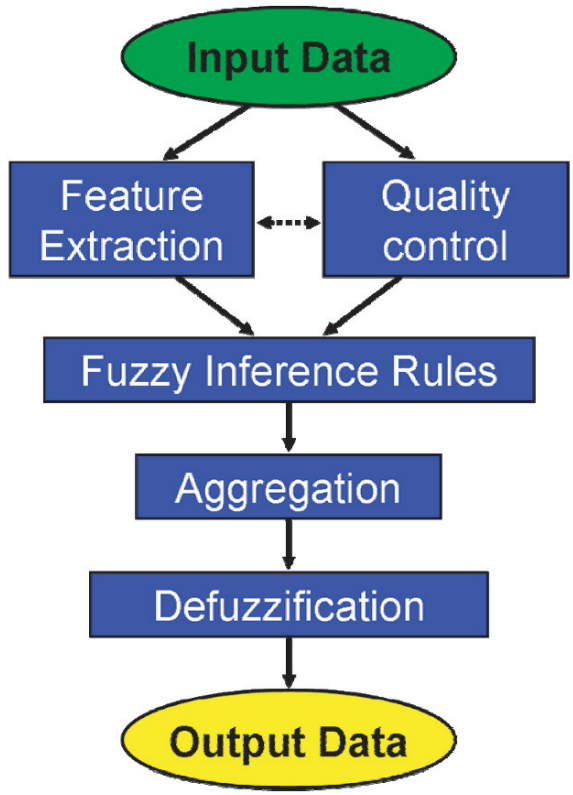


Fig. 6.17 Anatomy of a typical fuzzy logic algorithm.

6.10 Fuzzy Logic Algorithms

A typical fuzzy logic algorithm consists of several elements in addition to the fuzzy inference component (see Fig. 6.17). For instance, input data must often be pre-processed to derive desired features and perform appropriate quality control. Extracting features from the raw data may require complicated computations, such as convolving a kernel function with an image to identify boundaries, or computing contours, curvatures, derivatives or averages. These are then used to compute fuzzy set memberships or as inputs to interest maps. In conjunction with this process, input data should be quality controlled to ensure that corrupt data are censored or flagged with low confidence values so that their influence can be mitigated downstream. In environmental science applications, assessing data quality is often a vital component of a fuzzy logic algorithm that may even require a full-fledged fuzzy logic algorithm in its own right. After features and confidences have been computed, they may be used as input to fuzzy rules in the form of

Mamdami, Takagi-Sugeno, or fuzzy consensus reasoning. The outcomes of these rules are then aggregated and, if necessary, the result is "defuzzified" to produce a "crisp" output prediction or action. Fuzzy logic algorithms may also involve iterative components that run several times until a desired level of quality or certainty is achieved. And fuzzy logic algorithms may use combinations of different kinds of fuzzy reasoning, fuzzy numbers or even heuristics developed just for a particular problem; their common characteristic is that they mimic the human problem-solving approach of accommodating and exploiting ambiguity and postponing a "crisp" conclusion until the very last possible moment.

In the end, the fuzzy logic algorithm comprises a set of computations that provide a mapping from input data to an output prediction or action, and the methods described in this chapter are simply efficient ways for creating an appropriate mapping. If the human expert knowledge encoded in the fuzzy logic algorithm is of high quality and the algorithm is implemented correctly, the algorithm will usually do a good job on the problem it was designed to solve. On the other hand, if the human understanding of how to solve a given problem is inaccurate or incomplete, the fuzzy logic algorithm might not work as well as it potentially could.

One solution to this predicament is to use training data – pairs of input vectors and the associated ideal output, or "truth" values – to tune the fuzzy logic algorithm's parameters to optimize its performance. For instance, a training dataset might consist of the data used as input to a forecast algorithm along with subsequent measurements representing what actually happened. In this approach, the fuzzy logic algorithm is considered a function whose behavior can be modified by changing various parameters, which we may refer to collectively as the vector $\boldsymbol{\alpha}$. These might include values that control data preprocessing, parameters that describe each fuzzy set membership function (e.g., the vertices and values defining a piecewise-linear function), or the weights used in computing a fuzzy consensus. Indeed, tuning fuzzy logic algorithms is quite similar to training neural networks. In neural network case, the architecture of the network is defined by the number of hidden layers and nodes and the activation functions, and the parameters are the connection weights. Training occurs by modifying the weights to minimize the error between the neural network's outputs and the "truth" data, either by the gradient-descent backpropagation technique or some other method. A fuzzy logic algorithm has a different architecture, but it is still controlled by a set of parameters that can be adjusted to improve its performance (see Fig. 6.18). When fuzzy logic systems are optimized using training data, the result is sometimes called a *neuro-fuzzy system*. This approach to tuning a fuzzy logic system can be immensely powerful. It means that if a researcher has a good idea of what features are important and what the correct *form* (architecture) of an algorithm is but is not sure about some of the details – e.g., the ideal definition of each interest map – those details can be "filled in" using training data.

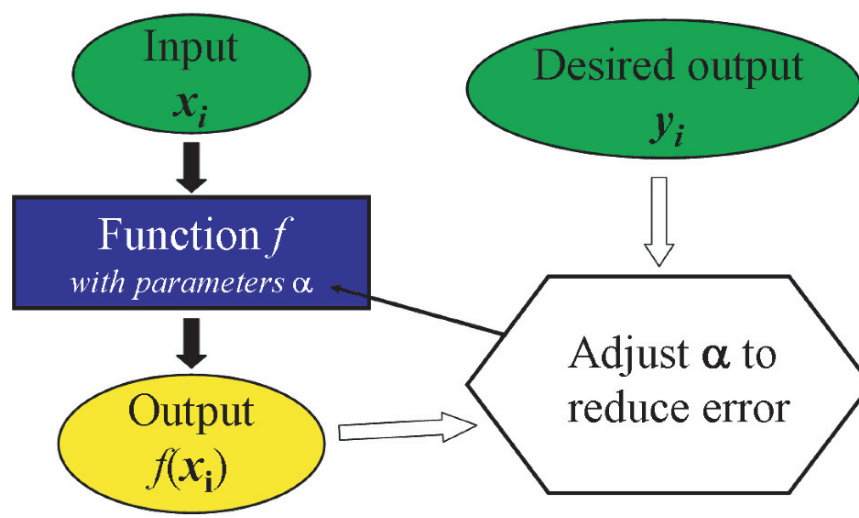


Fig. 6.18 Diagram illustrating how a fuzzy logic algorithm, represented as a function f determined by a set of parameters $\boldsymbol{\alpha}$, may be tuned to improve its performance when training data in the form of x_i, y_i pairs are available.

Several considerations might inform how the fuzzy logic algorithm is tuned. First, the error function (e.g., sum of squared differences between the predicted and target values) will be easy to write down in closed form only for the simplest algorithms, so a gradient-descent style optimization method would usually require estimating the gradient using multiple evaluations. However, for some fuzzy logic systems the error function may not be differentiable or even continuous in the parameters $\boldsymbol{\alpha}$, or it might have numerous local minima besides the global minimum that could trap a gradient descent technique. Furthermore, minimizing a simple error metric like the sum of squared errors is not necessarily what is desired for environmental science applications. Often, a forecaster might be interested in maximizing the True Skill Score or another skill statistic, and a remote sensing system might be evaluated based on the area under the Receiver Operating Characteristic (ROC) curve (see Chapter 2). Evaluating a decision support system might involve a complicated simulation of the costs and benefits associated with using the system. Such evaluation functions are in general not differentiable with respect to the fuzzy logic algorithm's parameters. Therefore, a general and effective way to tune a fuzzy logic algorithm is to use a genetic algorithm, representing the vector of parameters as a chromosome (see Chapter 5). Genetic algorithms do not make any assumptions about the differentiability or even continuity of the evaluation function and have been shown to work well for tuning fuzzy logic algorithms. The most difficult task may be to define an evaluation function that fully captures the salient details of performance, carefully treating issues of rare events, for instance. Sometimes, in fact, the evaluation function itself might best be implemented as a fuzzy logic algorithm that appropriately balances performance tradeoffs in different situations.

When a fuzzy logic algorithm is tuned "on line" as it is operating, the result is called an *adaptive fuzzy system*. For instance, the combination weights w_i in a Takagi-Sugeno system (6.4) or fuzzy consensus system (6.6) may be modified based on the recent performance of the associated predictive function, giving those functions that are doing well a bit more weight when they are corroborated and reducing the weight for those that poorly match the verification data. Since the dynamics of many natural systems tend to transition between distinct domains (e.g., as they orbit strange attractors), each of which has different characteristics or phenomenology, this capability allows the fuzzy algorithm to adapt appropriately as the situation changes. However, a fuzzy system that relies heavily on this sort of dynamic tuning may not perform well when the environmental system being measured transitions suddenly from one domain to another, and it may mask a problem in a data source that might best be dealt with directly. Whenever possible, it is probably preferable to identify the different domains or hidden variables that underlie the changing performance and incorporate them into the fuzzy logic algorithm itself. On the other hand, many environmental systems are quite complicated, and treating them with an adaptive fuzzy system may be the only practical approach.

6.11 Conclusion

Fuzzy logic provides a framework for creating expert systems that use information efficiently, encode human knowledge and heuristics, and are relative straightforward to implement. A central concept is that of fuzzy sets, which may be used to represent unsharp concepts like those commonly used in human communication and reasoning. A mathematical definition for fuzzy sets, accompanied by rules for manipulating them in analogy to classical sets, has been presented. A discussion of how fuzzy membership functions may be formed and combined via logical operations was followed by a description of Mamdami, Takagi-Sugeno and fuzzy consensus methods of inference. The fuzzy consensus method provides a basis for the confidence-weighted smoothing of data and the fuzzy k-nearest neighbor method for classifying data based on a collection of examples. Fuzzy clustering was presented as a way to discover structure in datasets that could be used as a step in developing a Takagi-Sugeno style fuzzy inference system. Fuzzy logic provides a natural way to integrate data quality control and information about measurement uncertainty into algorithms. When training data are available, the performance of a fuzzy logic algorithm can be optimized by using a genetic algorithm or another technique to tune the parameters governing its behavior, though a careful choice of the objective function is necessary to obtain good results. If tuning is done during algorithm operation as verification data become available, the resulting adaptive fuzzy system can maintain good performance despite

gradual changes in the environment or data sources. Fuzzy logic algorithms do not necessarily achieve the very optimum performance possible; rather, they fall into the category of "soft computing" methods that are robust, relatively easy to create and maintain, and perform very well on complex problems. These features make fuzzy logic a valuable tool for many environmental science prediction and decision support applications.

Acknowledgements Kent Goodrich first introduced me to fuzzy logic and is responsible for the story that begins this chapter, which describes how a project to produce weather hazard scores for flight service stations inadvertently resulted in a fuzzy logic algorithm. He, Jennifer Abernethy and Cathy Kessinger all read a draft of this chapter and offered helpful suggestions for improving it for which I am very grateful.

References

Albo, D. (1994). Microburst detection using fuzzy logic. *Terminal area surveillance system program project report* (49 pp.). Washington, DC: Federal Aviation Administration. [Available from the author at NCAR, P. O. Box 3000, Boulder, CO 80307.]

Albo, D. (1996). Enhancements to the microburst automatic detection algorithm. *Terminal area surveillance system program project report* (47 pp.). Washington, DC: Federal Aviation Administration. [Available from the author at NCAR, P. O. Box 3000, Boulder, CO 80307.]

Chi, Z., Hong, Y., & Tuan, P. (1996). *Fuzzy algorithms with applications to image processing and pattern recognition* (225 pp.). River Edge, NJ: World Scientific.

Delanoy, R. L., & Troxel, S. W. (1993). Machine intelligence gust front detection. *Lincoln Laboratory Journal*, *6*, 187–211.

Klir, G. J., & Folger, T. A. (1988). *Fuzzy sets, uncertainty and information* (355 pp.). Englewood Cliffs, NJ: Prentice Hall.

Klir, G. J., & Yuan, B. (Eds.). (1996). *Fuzzy sets, fuzzy logic and fuzzy systems: Selected papers by Lotfi A. Zadeh* (826 pp.). River Edge, NJ: World Scientific.

Mamdami, E. H. (1974). Applications of fuzzy logic algorithms for control of a simple dynamic plant. *Proceedings of IEEE*, *121*, 1585–1588.

McNiell, D., & Freiberger, P. (1993). *Fuzzy logic* (319 pp.). New York: Simon & Schuster.

Merritt, M. W. (1991). Microburst divergence detection for Terminal Doppler Weather Radar (TDWR). *MIT Lincoln Laboratory project report ATC-181* (174 pp). Lexington, MA: MIT Lincoln Laboratory.

Takagi, T., & Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Man and Cybernetics*, *15*, 116–132.

Tanaka, K. (1996). *An introduction to fuzzy logic for practical applications* (T. Tiimura, Trans., 138 pp.). New York: Springer.

Vivekanandan, J., Zrnic, D. S., Ellis, S. M., Oye, R., Ryzhkov, A. V., & Straka, J. (1999). Cloud microphysics retrieval using S-band dual polarization radar measurements. *Bulletin of American Meteorological Society*, *80*, 381–388.

Williams, J. K., & Vivekanandan, J. (2007). Sources of error in dual-wavelength radar remote sensing of cloud liquid water content. *Journal of Atmospheric and Oceanic Technology*, *24*, 1317–1336.

Yager, R. R., Ovchinnikov, S., Tong, R. M., & Nguyen, H. T. (Eds.) (1987). *Fuzzy sets and applications: Selected papers by L. A. Zadeh* (684 pp.). New York: Wiley.

Zadeh, L. H. (1965). Fuzzy sets. *Information and Control*, *8*, 338–353.

Zimmerman, H. J. (1996). *Fuzzy set theory and its applications* (435 pp.). Boston: Kluwer.

Missing Data Imputation Through Machine Learning Algorithms

7

Michael B. Richman, Theodore B. Trafalis, and Indra Adrianto

7.1 Introduction

How to address missing data is an issue most researchers face. Computerized algorithms have been developed to ingest rectangular data sets, where the rows represent observations and the columns represent variables. These data matrices contain elements whose values are real numbers. In many data sets, some of the elements of the matrix are not observed. Quite often, missing observations arise from instrument failures, values that have not passed quality control criteria, etc. That leads to a quandary for the analyst using techniques that require a full data matrix. The first decision an analyst must make is whether the actual underlying values would have been observed if there was not an instrument failure, an extreme value, or some unknown reason. Since many programs expect complete data and the most economical way to achieve this is by deleting the observations with missing data, most often the analysis is performed on a subset of available data. This situation can become extreme in cases where a substantial portion of the data are missing or, worse, in cases where many variables exist with a seemingly small percentage of missing data. In such cases, large amounts of available data are discarded by deleting observations with one or more pieces of missing data. The importance of this problem arises as the investigator is interested in making inferences about the entire population, not just those observations with complete data.

Before embarking on an analysis of the impact of missing data on the first two moments of data distributions, it is helpful to discuss if there are patterns in the missing data. Quite often, understanding the way data are missing helps to illuminate the reason for the missing values. In the case of a series of gridpoints, all gridpoints but one may have complete data. If the gridpoint with missing data is considered important, some technique to fill-in the missing values may be sought. Spatial interpolation techniques have been developed that are accurate in most situations (e.g., Barnes 1964; Julian 1984; Spencer and Gao 2004). Contrast this type of missing data pattern to another situation where a series of variables (e.g., temperature, precipitation, station pressure, relative humidity) are measured at a single location. Perhaps all but one of the variables is complete over a set of observations, but the last variable has some missing data. In such cases, interpolation techniques are not the logical alternative; some other method is required. Such problems are not unique to the environmental sciences. In the analysis of agriculture data, patterns of missing data have been noted for nearly a century (Yates 1933). Dodge (1985) discusses the use of least squares estimation to replace missing data in univariate analysis.

The majority of multivariate analysis techniques require that all variables be represented for each observation; hence, some action is required in the presence of missing data. In multivariate data sets, the patterns of missing data range from random to ordered. An example of the latter exists when some factor impacts a set of variables simultaneously (e.g., an ice storm

Michael B. Richman (✉)
School of Meteorology, University of Oklahoma, 120 David L. Boren Blvd, Suite 5900, Norman, OK 73072, USA
Phone: 405-325-1853; fax: 405-325-7689;
email: mrichman@ou.edu

Theodore B. Trafalis and Indra Adrianto
School of Industrial Engineering, University of Oklahoma, 202 West Boyd St., Room 124, Norman, OK 73019, USA
Emails: ttrafalis@ou.edu; adrianto@ou.edu

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

encrusting the anemometer blades for several sites). Traditionally, the most common method to address this type of missing data is with complete case deletion (Afafi and Elashoff 1966). This does have several advantages, including simplicity and all calculations are performed on the same observations. However, disadvantages arise from discarding information, including loss of precision and introduction of bias, in situations where the missing data are not exactly totally random. The loss of efficiency is due to increasing the variance of data after missing observations have been removed. The impact can be particularly insidious, for matrices with moderate to large numbers of variables. For example, in a network of 100 gridpoints, where each variable independently has a 1% chance of being missing, then the expected proportion of complete cases is $0.99^{100} = .366$. Therefore, only 36.6/.99 = 37% of the data values will be retained. With the availability of very large high-density global datasets with thousands of variables, even a tiny percentage of missing data on each variable can result in unexpectedly large fractions of the data removed by complete case deletion. The impact of such removal varies. For data that are independent, and with data that are missing completely at random, there is no correlation between variables and the issues of bias and variance inflation can be ignored (the assumption is that what remains is a perfect sample from the population and the variance can be adjusted by knowing the number of missing data) (Wilks 1932). However, such a situation is rarely encountered in meteorology, where sets of highly correlated variables are the norm. What occurs most often is an unknown increase in bias and inflation of the variance. A variation of patterns of missing data applies to certain observational networks (e.g., the National Climatic Data Center's Cooperative Observing Network for temperature and precipitation). These networks were established around 1900 and increased in density throughout much of the 20th century, had a maximum density in the 1970s and a decreasing density since that time. For a fixed data set based on these cooperative data (e.g., The Lamb-Richman datasets; Richman and Lamb 1985), temporal extensions of the network run into problems as long-existing stations drop out. Using such datasets for research requires special sampling strategies. One can partition such data sets into blocks of available stations, with successively smaller blocks as stations cease operation. Each block is analyzed separately to address missing data. Such techniques have been used successfully in the social sciences (Marini et al. 1980). Recalling the example of 1% missing data on a modestly sized analysis, it is apparent that with large data sets a possible outcome is that two or more variables may not share any observations. One variable may be present at times where another is missing and vice versa. This will make any partial associations between these variables meaningless.

In all the situations mentioned, the analyst must resolve how to proceed. There are several implicit assumptions that would suggest the analyst search for a value to impute or replace the missing data. The first assumption is that data that are missing represent values that are meaningful for subsequent analyses. This is the key reason why it is worthwhile to examine the efficacy of imputation methods. In meteorological analyses, sometime extreme weather conditions lead to instrument failures. Such information may be important to the analysis and suggests that the mechanism that gives rise to missing data may be important. If this is true, then the distribution may be worth investigating (Rubin 1976). The importance of this lies in the skewness of the distribution. More missing data may reside in the right or left tail of the distribution and replacement by random generation from a uniform or normal distribution could lead to bias.

In multivariate analyses, such as principal components, the data are reduced to a sample mean vector and a covariance matrix of the variables. Several options exist, including complete case analysis, inserting the sample mean for each piece of missing data and imputation of some modeled value. In the first case, a considerable amount of data may be discarded, and biased results emerge. The second option reduces the variances and covariances, leading to distortions in the eigenvalues and eigenvectors. Less is known about the third possibility. If the data can be assumed multivariate normal, the mean vector and covariance matrix can be estimated by maximum likelihood (Cox and Hinkley 1974). This involves replacing missing values on one variable by regressing it on a variable without missing data and calculating the predictions from the regression equation. A more general algorithm is based on expectation-maximization (EM) algorithm (Meng and Rubin 1991). The key to the EM algorithm is an iterative regression approach that converges to an appropriate value to impute. With the advent of nonlinear modeling techniques, such as

artificial neural networks (ANNs) and support vector regression (SVR), there is no reason to limit the iterative procedure to linear models.

The literature on the impacts of missing data in the atmospheric sciences, while not as extensive as that in statistics, is substantial. Despite this, several studies have noted decisions made to address missing data can have a profound impact on subsequent analyses (e.g., Kidson and Trenberth 1988; Duffy et al. 2001 summarize the importance of this issue). Additionally, proxy-based reconstruction methods are sensitive to the technique used to relate the data that are present to those that are missing (Rutherford et al. 2005; Mann et al. 2005). Climate monitoring issues impacted by gaps of missing data was reported by Stooksbury et al. (1999). For analysis of temperature observing networks, Kemp et al. (1983) compared several methods, finding regression yielded the smallest errors. However, using temperature records to track potential climate change has been hampered by missing data. In a study by Lund et al. (2001), only 359 of 1,221 stations in a network were used owing to missing data. Noting such problems, Lu et al. (2005) used more recent imputation methodology with results exhibiting smooth variations in gradients.

In cases where the individual observations are thought not important, deletion of every observation missing one or more pieces of data (complete case deletion) is common. This appears to be fostered by the wide use of data analysis packages (e.g., *nancov*, *isnan* in MATLAB Statistics Toolbox 2007) owing to the ease of its implementation. As the amount of missing data increases, tacit deletion can lead to bias in the first two statistical moments of the remaining data and inaccuracies in subsequent analyses. In datasets, where extreme values are of importance, extremes in wind speed and rainfall may be associated with meteorological conditions that lead to instrument failure and loss of data. Significantly, it is those extreme values that are of interest. If the data are deemed important to preserve, some method of filling in (or imputing) the missing values should be used.

Historically, if missing data were replaced, the statistical mean has been used most often as it was thought to minimize perturbations. Despite that, the use of the mean injects the same value into every instance of missing data and has been shown to create artificially low variation (Roth et al. 2005). What is desired is a principled method that uses information available in the remaining data to predict the missing values. Such techniques include substituting nearby data, interpolation techniques and linear regression using nearby sites as predictors. One class of technique that uses the information available to model uncertainty is known as iterative imputation.

The results from any technique used to estimate missing data depend, to a large extent, on the patterns of interrelated data (the degree of oversampling) and the manner in which the data are missing. The mechanism responsible for missing data should be assessed as random or systematic. In some cases, a few consecutive missing observations can be estimated with little error; however, if a moderate to large amount of data is missing, the results would be different. Motivated by such design questions, the present analysis seeks to examine how a number of techniques used to estimate missing data perform when various types and amounts of missing data exist. Imputation using EM algorithms have been applied to a wide range of problems (Meng and Pedlow 1992). The technique was developed in its present form by Hartley (1958) and further refined by Orchard and Woodbury (1972). The term EM was coined by Dempster et al. (1977). Several research studies have investigated various types of imputation. Rubin (1988) showed the improvements when using multiple imputation rather than single imputation. Variations of regression forms of imputation techniques have been applied to climate data (e.g., EM algorithm, Schneider 2001) with promising results.

In this work, different types of machine learning techniques, such as support vector machines (SVMs) and artificial neural networks (ANNs) are tested against standard imputation methods (e.g., multiple regression EM). These are compared to older techniques to document if they have the potential to offer improvement over older techniques. All methods are used to predict the known values of synthetic or randomly-generated data which have been thinned or altered to produce missing data. These data sets are on the order of 10 variables and 100 observations. Both linear and nonlinear synthetic data are used for comparison.

The data used in the analyses are described in 7.2. A brief overview of the methodology and experiments is provided in 7.3–7.4. The results are summarized in 7.5 and suggestions for implementation presented in 7.6.

7.2 Data Sets

Linear and nonlinear synthetic data are used in this study. Two linear synthetic data sets are randomly sampled from populations with two different correlations (0.3 and 0.7). These values span the range of correlation associated with the majority of geophysical data. A nonlinear synthetic data set is generated using the following function with 10 variables:

$$y = \frac{5\sin 10x_1}{10x_1} + 2x_2^3 - 3\cos 10x_3 + \frac{4\sin 5x_4}{5x_4} - 3\sin 4x_5 - 4x_6^2 - 9\frac{\sin 3x_7}{7x_7} + \frac{3}{\cos x_8} + 4\sin 3x_9 - 3x_{10}^4 + \zeta \quad (7.1)$$

where $-1 \le x_i < 1$ for $i = 1, \ldots, 10$, with a 0.02 increment, and ζ is the uniformly distributed noise between 0 and 0.5. The scatter plot of all pairs of variables used in the analyses, for each data set, can be found in Figs. 7.1–7.3. The degree of linearity (nonlinearity) can be assessed by visual examination of all pairs of variables.

All data sets consist of 100 observations with 10 variables. As missing data can occur in random or block patterns, both configurations are tested in this study. Each data set is altered to produce missing data by randomly removing one or more data and several blocks of data in two different percentages (10% and 40%) of the observations. In the case of blocks, the block size is 5% of the data set. For both percentages of missing data, multiple blocks of 5% are removed. In every case, since the data removed are known and retained for comparison to the estimated values, information on the error in prediction and the changes in the variance structure are calculated.

7.3 Methodology

The support vector machines (SVMs) and artificial neural networks (ANNs) are machine learning algorithms used in this research to predict missing data. Several standard methods such as casewise deletion, mean substitution, simple linear regression, and stepwise multiple regression, are employed for comparison.

7.3.1 Support Vector Machines

The SVM algorithm was developed by Vapnik and has become a favored method in machine learning (Boser et al. 1992). The version of SVMs for regression,

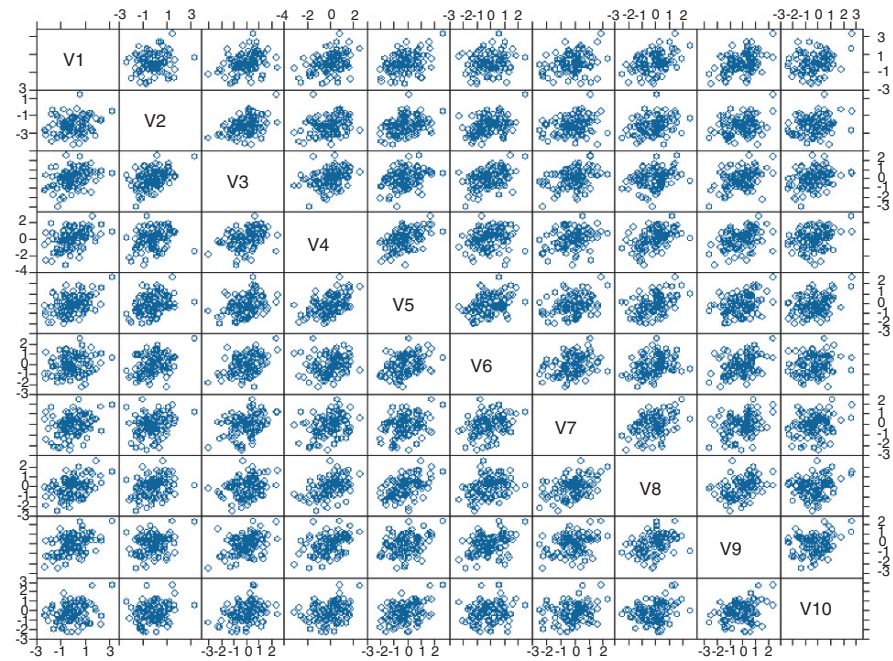


Fig. 7.1 A scatter plot matrix for the linear synthetic data set with the population correlation of 0.3. Each row or column represents a variable

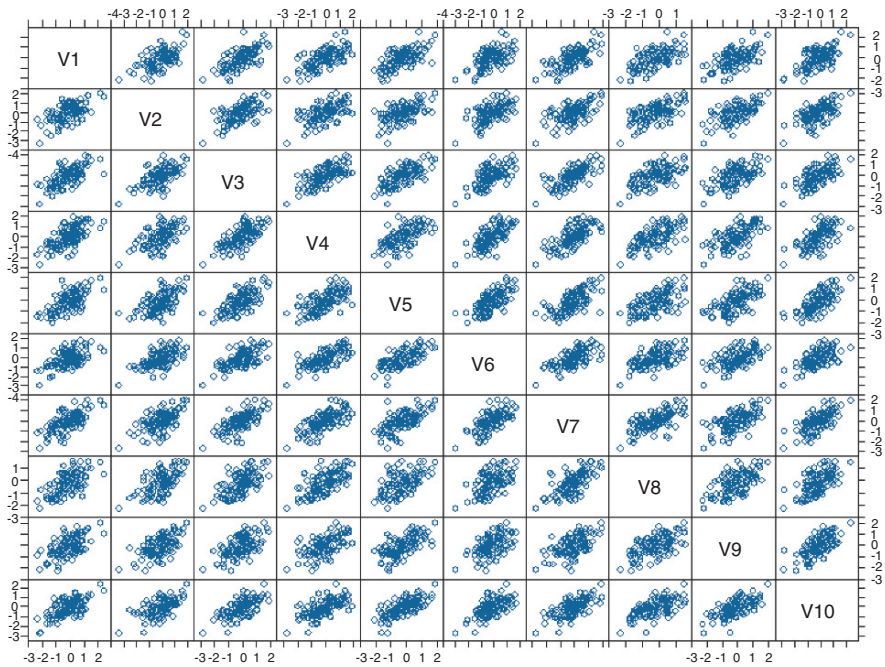


Fig. 7.2 A scatter plot matrix for the linear synthetic data set with the population correlation of 0.7. Each row or column represents a variable

called support vector regression (SVR), is used in this study. Trafalis et al. (2003) applied SVR for prediction of rainfall from WSR-88D radar and showed that SVR is more accurate, in terms of generalization error, than traditional regression.

The SVR formulation by Vapnik (1998) can be described as follows. Given a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell}$ of ℓ observations, our objective is to construct a function for approximating expected values $y : f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$ where $\mathbf{w}$ is the weight vector and b is a

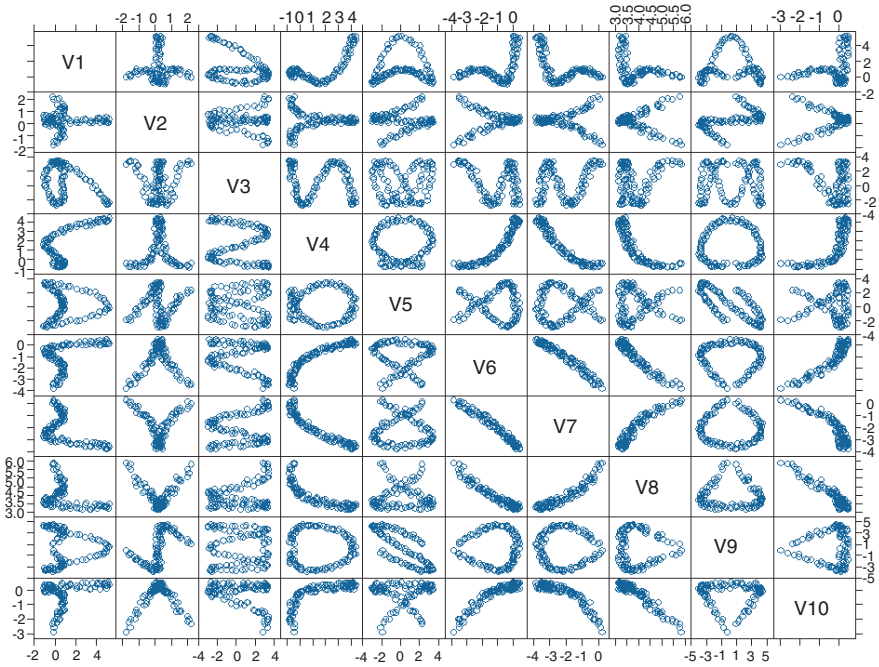


Fig. 7.3 A scatter plot matrix for the nonlinear synthetic data set. Each row or column represents a variable

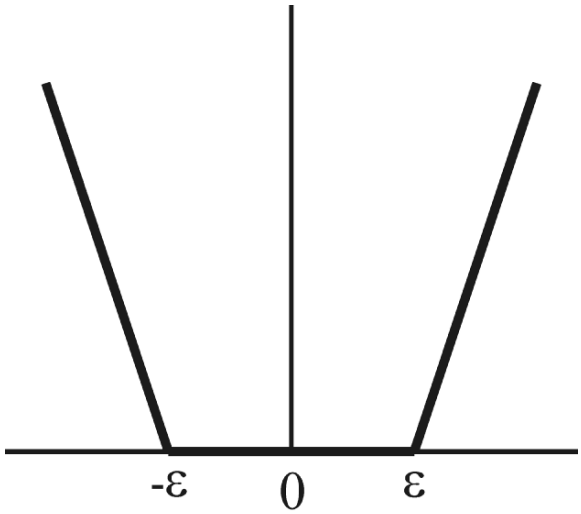


Fig. 7.4 The ε-insensitive loss function

bias. Vapnik (1998) proposed the linear ε-insensitive loss function in the support vector regression (SVR) formulation (Fig. 7.4). The linear ε-insensitive loss function is defined by:

$$L_\varepsilon(\mathbf{x}, y, f) = \begin{cases} 0 & \text{if } |y - f(\mathbf{x})| \le \varepsilon \\ |y - f(\mathbf{x})| - \varepsilon & \text{otherwise} \end{cases} \tag{7.2}$$

The SVR formulation can be represented as follows (Vapnik 1998):

$$\begin{aligned} &\min \phi(\mathbf{w}, \xi, \xi') = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i') \\ &\text{subject to } (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - y_i \le \varepsilon + \xi_i, \\ &y_i - (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \le \varepsilon + \xi_i', \\ &\xi_i, \xi_i' \ge 0, \quad i = 1, \ldots, l \end{aligned} \tag{7.3}$$

where $\mathbf{w}$ is the weight vector, b is a bias, C is a user-specified parameter, and ξ_i, ξ_i' are slack variables representing the deviations from the constraints of the ε-tube.

The SVR formulation in equation 7.3 can be transformed into the dual formulation using Lagrange multipliers α_i, α_i'. Note that $\mathbf{w} = \sum_{i=1}^{l}(\alpha_i' - \alpha_i)\mathbf{x}_i$. Using the linear ε-insensitive loss function, the dual formulation becomes (Vapnik 1998):

$$\begin{aligned} &\max Q(\alpha, \alpha') = \sum_{i=1}^{l} y_i(\alpha_i' - \alpha_i) - \varepsilon\sum_{i=1}^{l}(\alpha_i' + \alpha_i) \\ &\quad -\frac{1}{2}\sum_{i=1}^{l}\sum_{j=i}^{l}(\alpha_i' - \alpha_i)(\alpha_j' - \alpha_j)\langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle \\ &\text{subject to } \sum_{i=1}^{l}(\alpha_i' - \alpha_i) = 0, \\ &0 \le \alpha_i, \alpha_i' \le C, \quad i = 1, \ldots, l \end{aligned} \tag{7.4}$$

In the case of nonlinear problems (Fig. 7.5), if a function $\phi : \mathbf{x} \to \phi(\mathbf{x})$ exists, which maps $\mathbf{x}$ from the input space into a higher dimensional feature space, the inner product $\langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$ in equation 7.3 can be replaced by a kernel function $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \rangle$. The following kernel functions are used in this study:

1. Linear: $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$.
2. Polynomial: $k(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle + 1)^p$, p is the degree of polynomial.
3. Radial basis function (RBF): $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$, where γ is the parameter that controls the width of RBF.

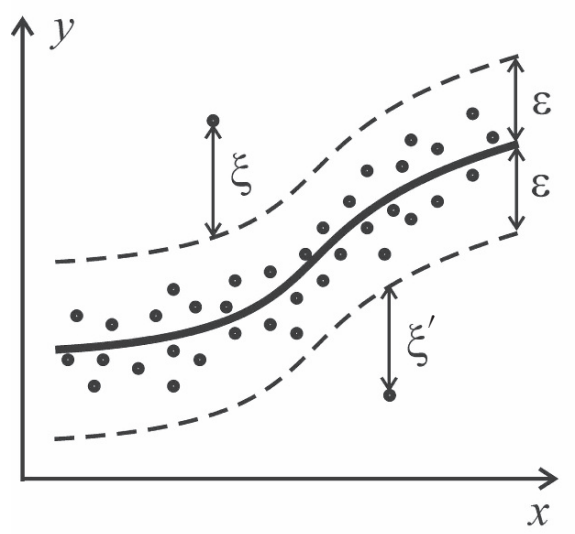


Fig. 7.5 Nonlinear regression problem

7.3.2 Artificial Neural Networks

Feedforward ANNs (Haykin 1999) are used with one hidden layer and different activation functions (linear and tangent-sigmoid) for the hidden and output layers. The scaled conjugate gradient backpropagation network is utilized for the training function. The formulation of feedforward ANNs is explained in Chapter 2.

7.3.3 Iterative Imputation

Fig. 7.6 shows the flowchart of the iterative imputation. The iterative imputation scheme used in this chapter can be described as follows:

Step 1. Given a data set of multiple observations (rows) and variables (columns). Identify which rows and columns have missing data.

Step 2. Separate the observations that do not contain any missing data (*set 1*) from the observations that have missing data (*set 2*).

Step 3. *Initialization (Iteration 1)*. Suppose we have n missing data. For each column in *set 2* that has missing data, construct a regression function using *set 1*. The dependent or response variable is the column that has missing data and the independent or predictor variables are the other columns. Predict the missing data for each column in *set 2* using the corresponding regression function. Therefore, we create values v_i^1, where $i = 1, \ldots, n$, to impute the missing data in *set 2*.

Step 4. *Iteration 2*. Merge the imputed set from the previous step with *set 1*. For each column in *set 2* that has missing data, construct again a regression function using this merged set. Predict the missing data for each column in *set 2* using the new corresponding regression function. In this iteration, we recreate values v_i^2, where $i = 1, \ldots, n$, to impute the missing data in *set 2*.

Step 5. Calculate the difference (δ) between the predicted values from the current iteration (v_i^k) with the ones from the previous iteration (v_i^{k-1}) for all $i = 1, \ldots, n$, where k is a positive integer: $\delta = \frac{1}{n} \sum_{i=1}^{n} \left| v_i^k - v_i^{k-1} \right|$.

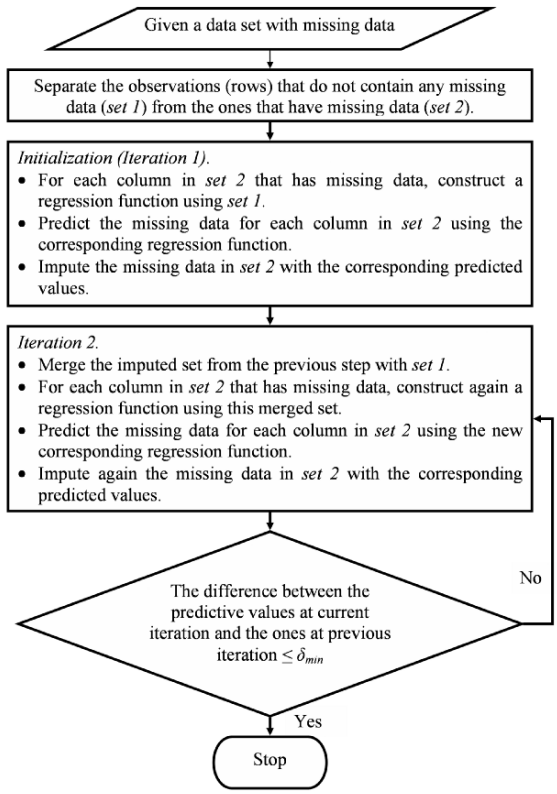


Fig. 7.6 A flowchart of the iterative imputation

Step 6. If $\delta \leq \delta_{\min}$, stop the iteration, where $\delta_{\min}$ is a user-defined small positive value, otherwise go to Step 7.

Step 7. *Iteration 3*. The same as *Iteration 2*.

Several iterations can be applied to construct imputed data sets. The iteration should be stopped when the difference between the predictive values at the current iteration and the ones at the previous iteration $\leq \delta_{\min}$. In this study, we apply SVR, ANNs, and stepwise-regression to construct the regression functions for iterative imputation methods. For non-iterative imputation methods, we perform mean substitution and simple linear regression.

In order to document the accuracy of our methods to estimate the first statistical moment of our data, we use the mean squared error (MSE) to measure the difference between the actual values from the original data set and the corresponding imputed values. For n missing values, the MSE is the average squared error

between the actual values y_i and the imputed values v_i for $i = 1, \ldots, n$. The MSE formulation can be written as: $MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - v_i)^2$.

We cannot use the MSE to measure the performance of casewise deletion because there is not any imputation to replace missing data. However, we can compare the variance and covariance structures of the original data set to the imputed data set with the casewise deletion. The difference of variance between the original data set and the imputed data set is measured using the mean absolute error (MAE). For m columns with missing values the MAE is the average absolute error between the variance of variables in the original data set s_i and the variance of variables in the imputed data set t_i for $j = 1, \ldots, m$. The MAE formulation for the variance difference can be expressed as: $MAE = \frac{1}{m} \sum_{j=1}^{m} |s_j - t_j|$. Furthermore, the difference between the lower triangle of the covariance matrix of the original data set and the one of the imputed data set is also measured using the MAE. Suppose q is the number of elements the lower triangle of the covariance matrix, the MAE is the average absolute error between the lower triangle elements of the covariance matrix of the original data set c_j and the ones of the imputed data set d_j for $j = 1, \ldots, q$. The MAE formulation for the lower triangle of the covariance matrix difference can be expressed as: $MAE = \frac{1}{q} \sum_{j=1}^{q} |c_j - d_j|$.

7.4 Experiments

For each data set (10%, 40%, blocks of 5% missing data), we resample 100 times to obtain stable statistics and confidence intervals. The locations of missing data are assigned randomly or in blocks of missing data for five consecutive observations in several locations at one column. Then we applied the imputation methodology as described in 7.3 to predict missing data where SVR, ANNs, and stepwise-regression are used. For these experiments, three iterations are applied for each method. Additionally, the same data sets are used for non-iterative imputation methods using mean substitution and simple linear regression to substitute missing data.

The experiments are performed in the MATLAB environment. The SVR experiments use LIBSVM toolbox (Chang and Lin 2001) whereas the ANN, simple linear and stepwise regression experiments utilize the neural network and statistics toolboxes, respectively.

7.5 Results

7.5.1 Results for the Observations with Random Missing Data

Table 7.1 and Fig. 7.7 show the MSE results for each method with two different percentages of the observations that have one or more missing data for the linear synthetic data sets. The average MSE from 100 different randomly seeded data sets is reported as well as bootstrapped resampled confidence intervals. For SVR experiments, different combinations of kernel functions (linear, polynomial, radial basis function) and C values are applied to determine the parameters that give the lowest MSE. After experimentation, the "best" SVR parameters use the linear kernel and ε-insensitive loss function with $\varepsilon = 0.3$, and C = 0.05. For ANNs, we train several feed-forward neural networks using one hidden layer with different number of hidden nodes (from 1 to 10) and different activation functions (linear and tangent-sigmoid) for the hidden and output layers. The scaled conjugate gradient backpropagation network is used for the training function. To avoid overfitting, the training stops if the number of iterations reaches 100 epochs or if the magnitude of the gradient is less than 10^{-6}. The neural network that gives the lowest MSE has one hidden node with the linear activation function for both hidden and output layers. For stepwise regression, the maximum p-value that a predictor can be added to the model is 0.05 whereas the minimum p-value that a predictor should be removed from the model is 0.10. For mean substitution, the missing data in a variable are replaced with the mean of its variable. Simple linear regression uses only one independent variable that has the highest correlation with the response variable to predict missing data. Only a single solution (no additional iterations) is used for simple regression.

Results for this experiment for an underlying population correlation of 0.3, where the amount of missing data are 10%, show that substitution of the mean is worst in terms of recovering the true information removed from the analysis (Table 7.1; Fig. 7.7a).

Table 7.1 The average MSE for all methods with 10% and 40% of the observations missing for the linear data sets with 0.3 and 0.7 correlations

Correlation	% of the observations missing	Iteration	SVR	Stepwise reg.	ANN	Mean subst.	Simple lin. reg.
		1	0.766	0.850	0.791	1.022	0.911
	10%	2	0.754	0.832	0.784		
0.3		3	0.754	0.829	0.784		
		1	0.821	0.904	0.865	1.028	0.934
	40%	2	0.787	0.847	0.820		
		3	0.787	0.849	0.823		
		1	0.345	0.404	0.365	0.873	0.533
	10%	2	0.342	0.396	0.363		
0.7		3	0.342	0.398	0.364		
		1	0.362	0.419	0.386	0.826	0.511
	40%	2	0.352	0.398	0.370		
		3	0.353	0.405	0.373		

The bootstrap resampling indicates the substitution of the mean model is statistically different from a noniterative simple regression (i.e., the bootstrapped confidence intervals on the mean substitution do not overlap the mean of that based on simple linear regression). Of the iterative techniques applied, the support vector regression has the smallest MSE, though it was statistically indistinguishable from either ANN or stepwise regression by the third iteration. For these data, the SVR, stepwise regression and ANN had lowest errors after two iterations. As the amount of missing data was increased to 40%, the mean substitution was clearly the most inaccurate imputation technique, followed by simple linear regression (Table 7.1; Fig. 7.7b). The three iterative techniques are all clearly superior to the aforementioned techniques by the second iteration where the minimum MSE was achieved, and the confidence intervals do not overlap either noniterative method.

The next experiment increased the population correlation structure to 0.7. In theory, this should result in more accurate imputations as there is more linear structure among the data. This can be seen clearly by comparing the results for 10% missing data (Table 7.1; Fig. 7.7c) to those for a lower population correlation (Table 7.1; Fig. 7.7a). In the experiment with 0.7 population correlation, the MSE for mean imputation decreases by over 10% whereas, for the simple regression it decreases about 40%. The best results can be seen clearly for the iterative imputations where the confidence intervals do not overlap the means of the noniterative techniques (Fig. 7.7c). When the amount of missing data is increased, results (Table 7.1; Fig. 7.7d) are consistent to the 10% experiment (Fig. 7.7c) with a slight increase in MSE for every technique due to the additional missing data. At 40% missing data (Table 7.1; Fig. 7.7d), these trends toward slightly larger MSE continue but the hierarchy of three distinct levels of accuracy remain unchanged (mean substitution worst, simple linear regression, iterative imputation best). In all cases, the iterative imputation techniques are most accurate and two iterations result in convergence of the MSE values. When there is a large correlation among the variables in an analysis, all the iterative imputation techniques are much more accurate than using the mean. This is expected since, if the population correlation were zero, then the best regression would be the mean.

The impact of casewise deletion and the various imputation techniques on the variance and covariance structure was investigated. As expected, casewise deletion had the most noteworthy negative impact on the true variance and covariance (Fig. 7.8) at all percentages of missing data for both 0.3 and 0.7 correlations. Unlike the MSE errors in the replaced data elements (Fig. 7.7), which were modest within any technique, the impact of increasing the amount of missing data on the variance and covariance structures is obvious (compare Fig. 7.8a, b or 7.8c, d) with no overlap in the bootstrapped confidence intervals. Of the imputation techniques, substituting the mean had the expected result of underestimating the true

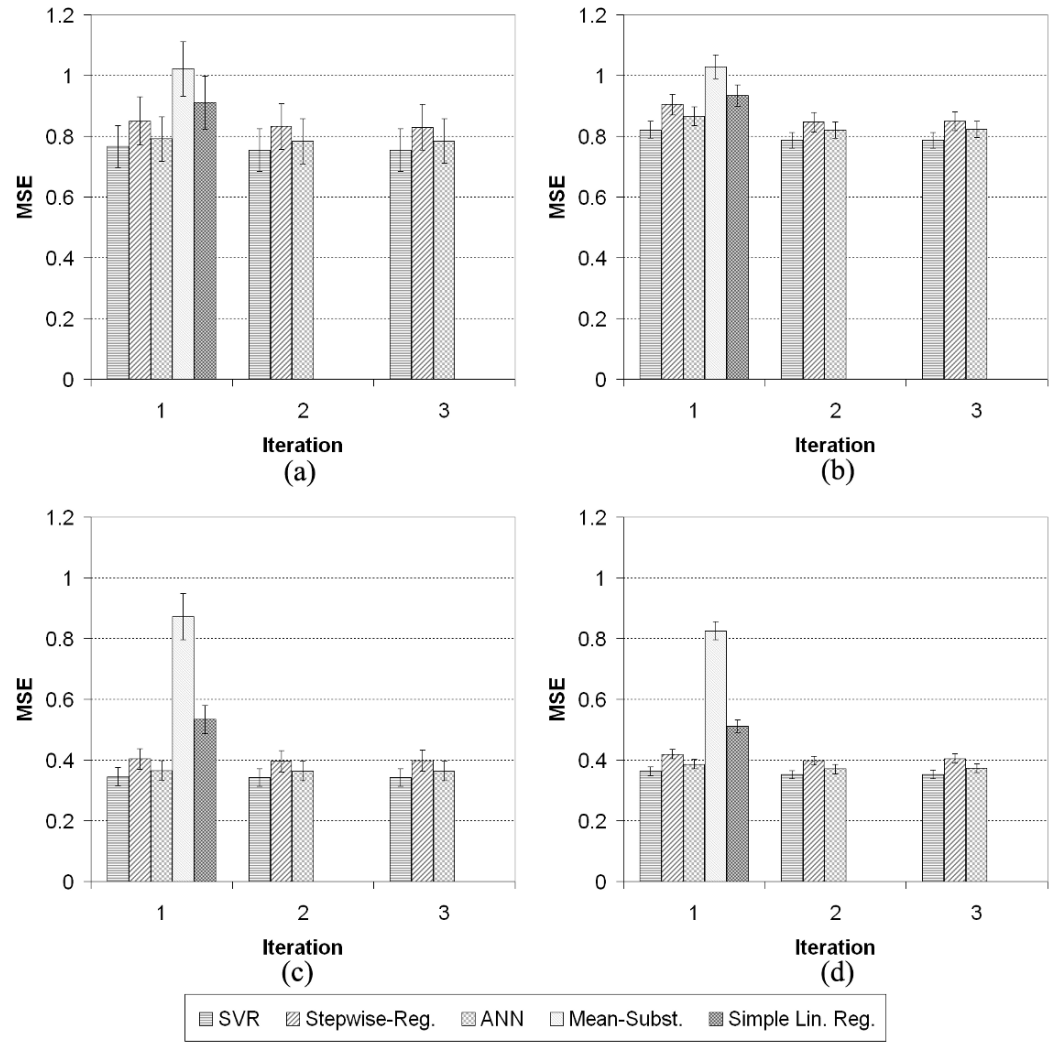


Fig. 7.7 The average MSE for all methods after three iterations with 95% confidence intervals when (**a**) 10% of the observations missing for 0.3 correlation, (**b**) 40% of the observations missing for 0.3 correlation, (**c**) 10% of the observations missing for 0.7 correlation, and (**d**) 40% of the observations missing for 0.7 correlation

variance and covariance (Fig. 7.8) for both 0.3 and 0.7 correlations. The remaining techniques were considerably more accurate in coming closer to the true variance structure. The iterative techniques were statistically indistinguishable as their confidence intervals overlap.

For the variance structures (Fig. 7.8a, b), casewise deletion leads increased error by a factor of 2.5 to 3 over the iterative techniques. This is highly significant and considerably worse than the second most inaccurate method, mean substitution. As the percentage of missing data increases, the iterative techniques emerge with lower MAE compared to the noniterative techniques or casewise deletion. For the covariance structures (Fig. 7.8c, d), the difference is significant at all percentages of missing data and the differential between the casewise deletion and the remaining techniques increases. *It is clear how deleterious casewise deletion is for variance/covariance reconstruction.*

The same experiments are applied for the nonlinear synthetic data set. Table 7.2 and Fig. 7.9 illustrate the results for each method with four different percentages

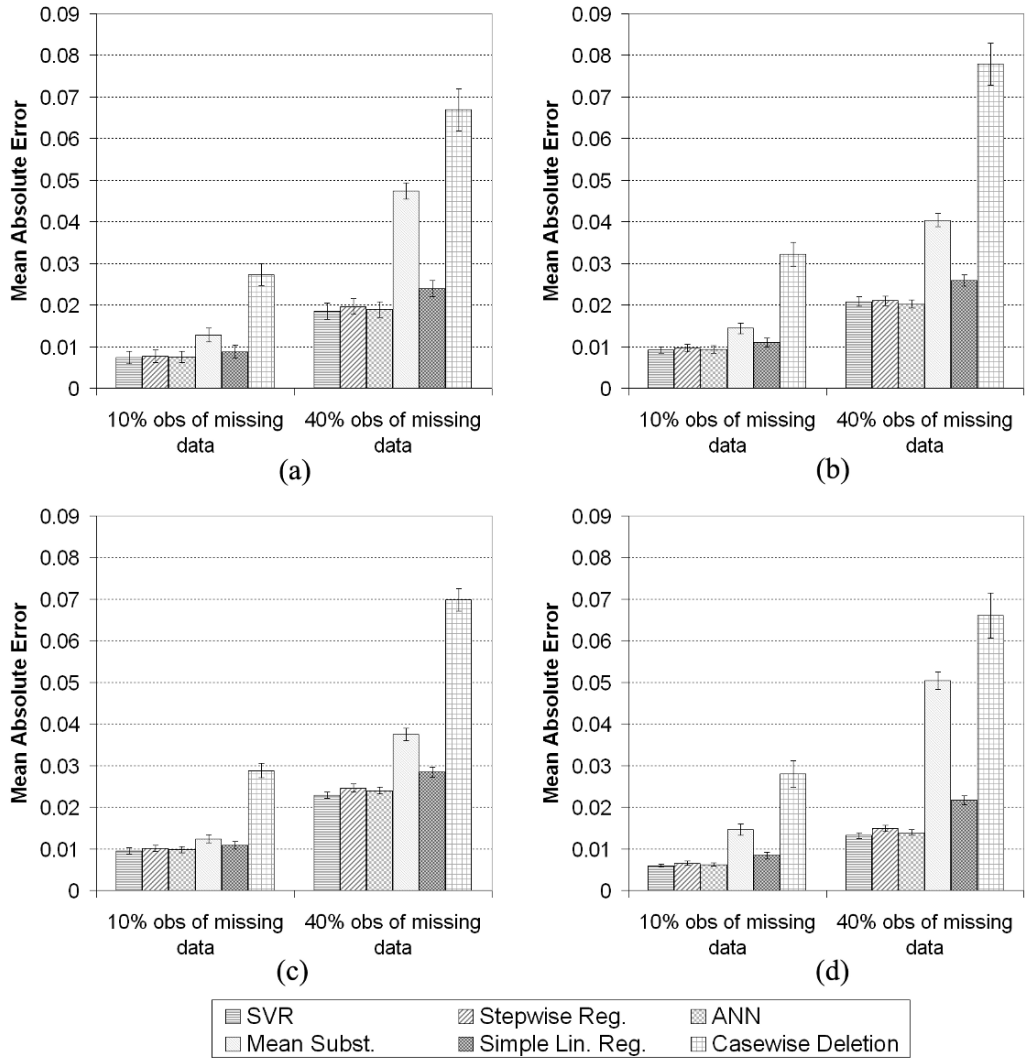


Fig. 7.8 Bar charts with 95% confidence intervals illustrating the difference of variance between the original and imputed data sets with (**a**) 0.3 and (**b**) 0.7 correlations. Bar charts with 95% confidence intervals illustrating the difference between the lower triangle of the covariance matrix of the original data set and the one of the imputed data set with (**c**) 0.3 and (**d**) 0.7 correlations

of the observations that have one or more missing data. Different combinations of kernel functions and C values are applied for SVR experiments to choose parameters that give the lowest MSE. The best SVR parameters use the radial basis function kernel and ε-insensitive loss function with $\gamma = 0.2$, $\varepsilon = 0.1$, and $C = 10$. Also, we train several feed-forward neural networks using one hidden layer with different number of hidden nodes (from 1 to 10) and different activation functions (linear and tangent-sigmoid) for the hidden and output layers for these nonlinear data. The scaled conjugate gradient backpropagation network is used for the training function. The training ends if the number of iterations reaches 100 epochs or if the magnitude of the gradient is less than 10^{-6}. Owing to the nonlinearity, the best neural network with the lowest MSE has four hidden nodes with the tangent-sigmoid activation function for the hidden layer and the linear activation function for the output layer. For stepwise regression, mean substitution, and simple linear regression, the same procedure was used as for the linear data sets.

Table 7.2 The average MSE for all methods with 10% and 40% of the observations missing for the nonlinear data set

% of the observations missing	Iteration	SVR	Stepwise reg.	ANN	Mean subst.	Simple lin. reg.
10%	1	0.010	0.035	0.014	0.362	0.142
	2	0.009	0.034	0.012		
	3	0.009	0.034	0.011		
40%	1	0.046	0.086	0.044	0.362	0.152
	2	0.017	0.045	0.022		
	3	0.015	0.044	0.018		

Results for the nonlinear experiment at 10% missing data (Table 7.2; Fig. 7.9a) were fundamentally different than for the linear analyses (Table 7.1; Fig. 7.7a). The mean substitution had errors well over an order of magnitude larger than the iterative imputation techniques. Furthermore, the simple linear regression had the second worst MSE in this experiment. The most striking difference was that both ANN and SVR had errors that were about one third those for stepwise regression and the bootstrapped confidence intervals for ANN and SVR did not overlap those of stepwise regression (the traditional EM approach). The iteration number had no significant impact on the MSE in these experiments. As the amount of missing data was increased to 40%, the MSE increased dramatically for the iterative imputation techniques and remained relatively constant for the mean substitution and simple regression (Table 7.2; Fig. 7.9b). *Despite that, the iterative techniques were indisputably more accurate than either the mean substitution or the simple regression*. ANN and SVR iterative imputation were the most accurate of all the techniques tested. The same behavior can be seen for the variance reconstructions (Fig. 7.10a) and the covariance MAE (Fig. 7.10b). *In both cases, the best imputation techniques (ANN and SVR) had errors close to an order of magnitude less than casewise deletion*.

7.5.2 Results for the Observations with Blocks of Missing Data

Table 7.3 and Fig. 7.11 illustrate the results for each method with two different percentages of the

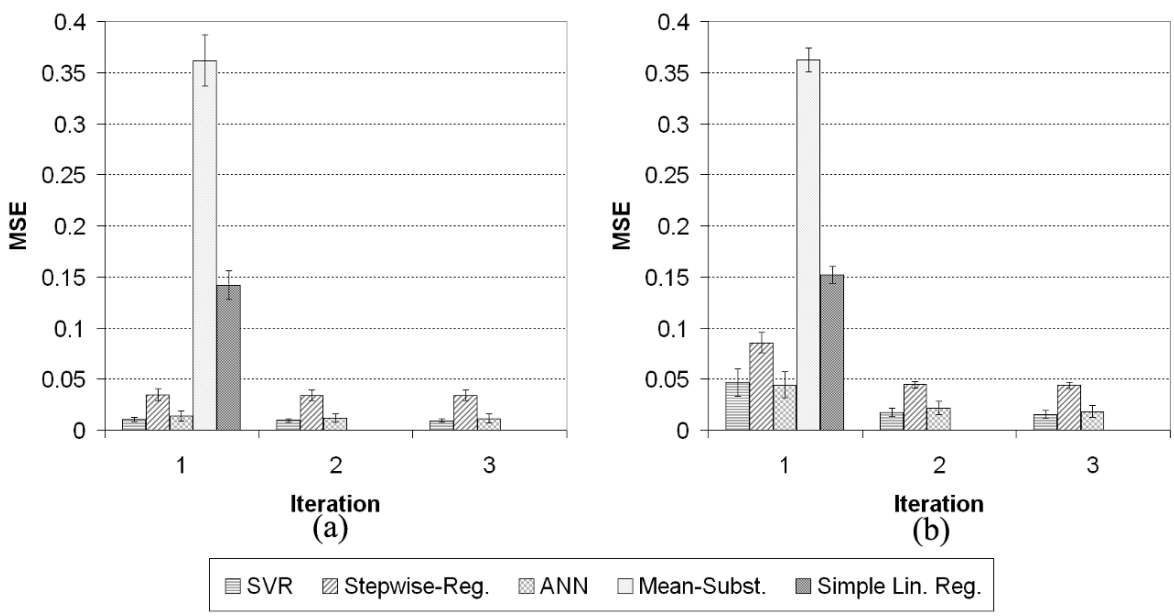


Fig. 7.9 The average MSE for all methods after three iterations with 95% confidence intervals when (**a**) 10% and (**b**) 40% of the observations missing for the nonlinear data set

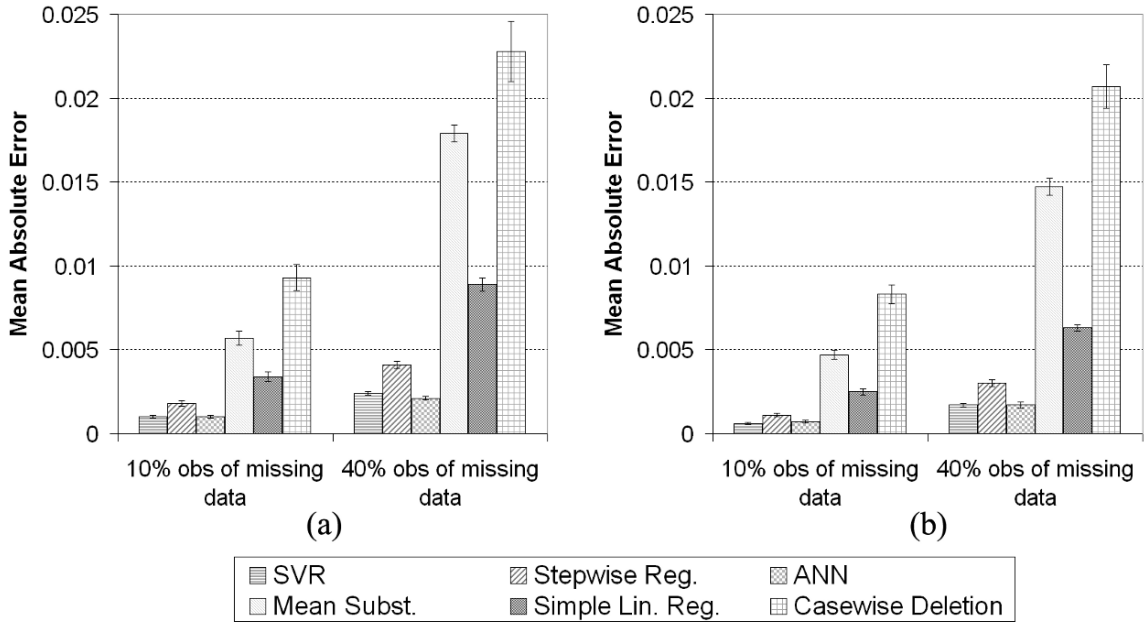


Fig. 7.10 (**a**) A bar chart with 95% confidence intervals illustrating the difference of variance between the original and imputed nonlinear data set for random missing data. (**b**) A bar chart with 95% confidence intervals illustrating the difference between the lower triangle of the covariance matrix of the original nonlinear data set and the one of the imputed nonlinear data set for random missing data

observations that have blocks of missing data for the nonlinear synthetic data sets. The same settings used in the random missing data experiments for the nonlinear data sets are employed for all methods in this experiment. Results for 10% missing data (Table 7.3; Fig. 7.11a) indicated that the mean substitution had an order of magnitude larger MSE than stepwise regression. The simple linear regression had about four times the error of stepwise regression. ANN and SVR, both have errors about one fourth those of stepwise regression, indicating the improvement that can be gained when switching to modern machine learning methods. At 40% missing data, the errors for ANN and SVR are about the same (Table 7.3; Fig. 7.11b). In contrast, the MAE results for the variance are relatively similar for all imputation techniques at 10% missing data (Fig. 7.12a) and much larger for casewise deletion (confidence intervals do not overlap). As the percentage of missing data increase from 10% to 40%, the negative impact on the variance of using the mean can be seen. For the covariance results (Fig. 7.12b), the insertion of the mean no longer had a negative impact on for the nonlinear data with block removal but rather had the lowest MAE by a slight amount for 10% and 40% missing data. This can be explained by visualizing a nonlinear function with a substantial proportion of the pattern removed, so much that the original pattern is not recognizable to train. It is obvious that the data requirements to fit the proper nonlinear surface to data are considerably higher than to fit a linear feature

Table 7.3 The average MSE for all methods with 10% and 40% of the observations have blocks of missing data for the nonlinear data set

% of the observations missing	Iteration	SVR	Stepwise reg.	ANN	Mean subst.	Simple lin. reg.
10%	1	0.011	0.043	0.012	0.366	0.136
	2	0.011	0.042	0.011		
	3	0.011	0.041	0.011		
40%	1	0.011	0.038	0.015	0.342	0.124
	2	0.011	0.038	0.013		
	3	0.011	0.038	0.013		

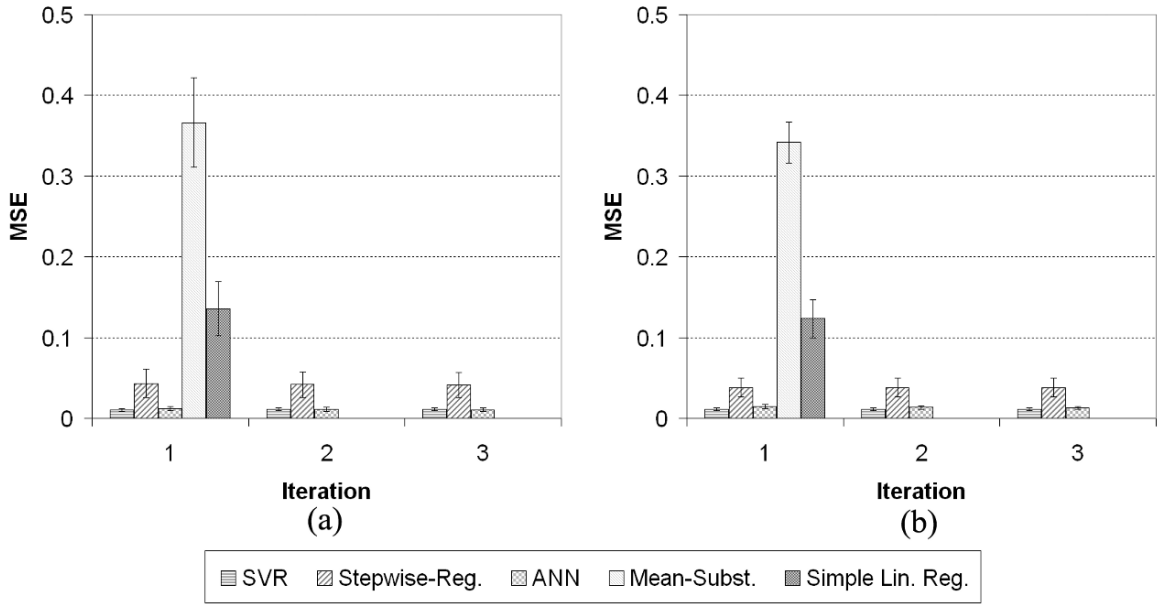


Fig. 7.11 The average MSE with 95% confidence intervals for all methods with (**a**) 10% and (**b**) 40% of the observations have blocks of missing data for the nonlinear data set

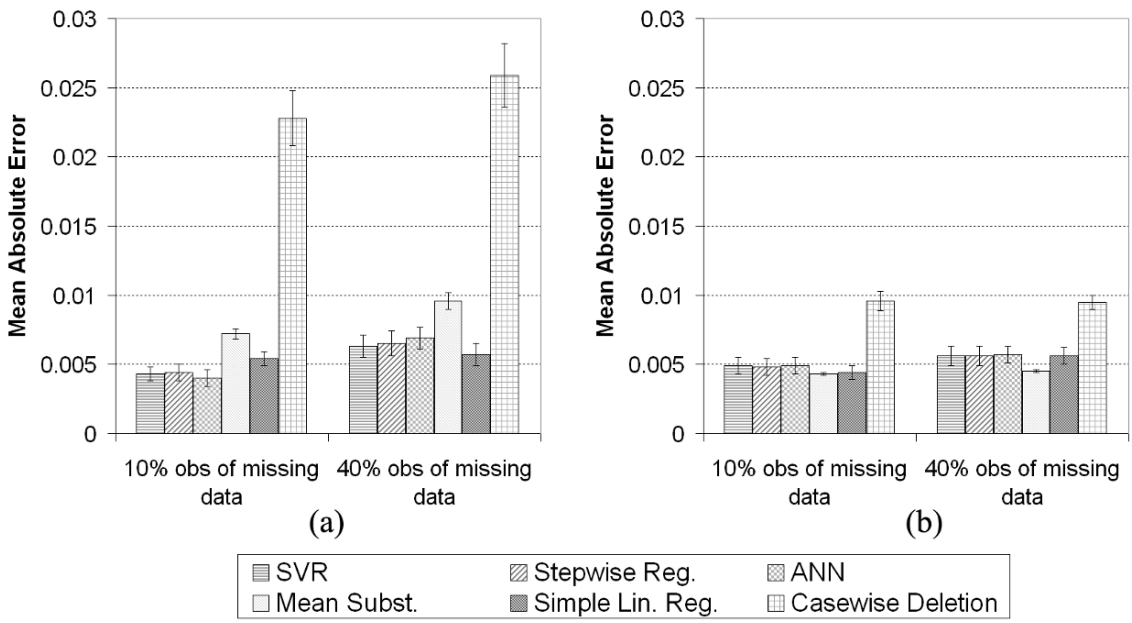


Fig. 7.12 (**a**) A bar chart with 95% confidence intervals illustrating the difference of variance between the original and imputed nonlinear data set for blocks of missing data. (**b**) A bar chart with 95% confidence intervals illustrating the difference between the lower triangle of the covariance matrix of the original nonlinear data set and the one of the imputed nonlinear data set for blocks of missing data

or to use the mean value. *In all cases casewise deletion continued to cause highest MAE in the variance and covariance structures*.

7.6 Conclusions and Actionable Recommendations

Linear and nonlinear synthetic data sets are tested to determine the impact of removing data in random and block configurations on imputed values and the variance-covariance structure of the datasets. Elements are removed from these data matrices randomly in increments of 10% and 40%. Casewise deletion, mean substitution, simple regression, and imputation with stepwise linear regression, ANN and SVR, are tested to determine how well the techniques can estimate the missing values (except for the casewise deletion where it is impossible to estimate the missing values).

Results of extensive experimentation with the aforementioned methods provide interesting findings and implications. By estimating the missing data and then comparing these estimates with the known values, the amount of signal that can be recovered is identified. In all experiments, the use of casewise deletion causes large errors in the variance and covariance of the estimates. For data that have largely linear relationships, the use of mean substitution leads to large errors in the imputed values and in the variance estimates. Simple linear regression is a minor improvement over use of the mean. The lowest errors are found for the iterative imputation methods. Among the imputation techniques tested, ANN and SVR are ranked lowest in data error reported. Experiments testing the impact of the type of pattern of missing data (random versus block) suggest generally a similar behavior between the two types of missing data, although the accuracy was usually lower for block removal of data. Since learning methods were nearly always superior to multiple linear regression, more widespread use of ANN and SVR over traditional EM techniques is warranted in situations when it is important to obtain accurate estimates of missing data.

The investigator desiring to analyze a data set containing missing values can be guided by the results of this study. Small amounts of missing data, in the 1–5% range, had a measurable impact on the results in a more extensive study (not shown herein, but similar to those results shown for 10% missing data). In an applied setting, with a dataset containing missing values, there are no known values for the missing data. Given the poor performance, across all of the experiments, of casewise deletion in distorting both the variance and covariance structures of the data, that approach cannot be recommended. Similarly, under most circumstances, imputation using the mean value leads to poor estimates of the missing data, underestimation of the variance and moderate errors in the covariances. In general, that method cannot be recommended. Synthesizing the findings based on the remaining techniques reveals that understanding the types of relationships among the data is a prerequisite to using the results presented. If the relationships are roughly linear, and if the covariance structure among the sample variables is not independent, then an iterative imputation technique can be recommended. For this type of data configuration, iterative imputation based on an EM algorithm was nearly as accurate as nonlinear machine learning methods, such as ANN and SVR. The stronger the relationships among the variables, and the fewer missing data, our results showed the more accurate the imputed value. The linearity can be assessed visually through the use of biplots or scatterplots. In the special case of few missing data with strong covariances between the variables, simple (non-iterative) regression may suffice if the analyst is willing to accept somewhat higher errors in the imputed values. For this case only, the variance estimates given by simple regression are fairly accurate. However, iterative multiple regression is more accurate than simple regression.

If the data are nonlinearly related or of unknown relationship, this research shows that iterative imputation using machine learning algorithms is superior to all other methods. Once again, casewise deletion will lead to errors in estimation of the variance and covariance structures of an order of magnitude higher than machine learning techniques. If the actual missing values are needed with nonlinear relationships, the worst method was substituting the mean, followed closely by simple linear regression. Of the iterative techniques, we recommend either ANN or SVR over multiple regression, as the errors are reduced by over 50% compared to EM methodology.

If the missing data are not randomly spaced but, rather, missing in blocks, the recommendation is to use a machine learning algorithm. The errors associated with the EM method are about three times larger than either ANN or SVR. Both noniterative simple regression and mean substitution lead to errors of an order of magnitude or more than the machine learning algorithms. The use of casewise deletion leads to massive errors in the variance and covariance estimation, whereas simple linear regression leads to higher errors in the variance field.

Acknowledgements Funding for this research was provided under NOAA Cooperative Agreement NA17RJ1227 and National Science Foundation Grants ATM-0527934 and EIA-0205628.

References

Afafi, A. A., & Elashoff, R. M. (1966). Missing observations in multivariate statistics: Review of the literature. *Journal of the American Statistical Association*, *61*, 595–604.

Barnes, S. L. (1964). A technique for maximizing details in numerical weather map analysis. *Journal of Applied Meteorology*, *3*, 396–409.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In D. Haussler (Ed.), *5th Annual ACM Workshop on COLT* (pp. 144–152). Pittsburgh, PA: ACM Press.

Chang, C., & Lin, C. (2001). LIBSVM: A library for support vector machines. From http://www.csie.ntu.edu.tw/~cjlin/libsvm

Cox, D. R., & Hinkley, D. V. (1974). *Theoretical statistics*. New York: Wiley.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, *B*39, 1–38.

Dodge, Y. (1985). *Analysis of experiments with missing data*. New York: Wiley.

Duffy, P. B., Doutriaux, C., Santer, B. D., & Fodor, I. K. (2001). Effect of missing data estimates of near-surface temperature change since 1900. *Journal of Climate*, *14*, 2809–2814.

Hartley, H. O. (1958). Maximum likelihood estimation from incomplete data. *Biometrics*, *14*, 174–194.

Haykin, S. (1999). *Neural networks: A comprehensive foundation* (2nd ed.). Englewoods Cliffs, NJ: Prentice-Hall.

Julian, P. R. (1984). Objective analysis in the tropics: A proposed scheme. *Monthly Weather Review*, *112*, 1752–1767.

Kemp, W. P., Burnell, D. G., Everson, D. O., & Thomson, A. J. (1983). Estimating missing daily maximum and minimum temperatures. *Journal of Applied Meteorology*, *22*, 1587–1593.

Kidson, J. W., & Trenberth, K. E. (1988). Effects of missing data on estimates of monthly mean general circulation statistics. *Journal of Climate*, *1*, 1261–1275.

Lu, Q., Lund, R., & Seymour, L. (2005). An update on U.S. temperature trends. *Journal of Climate*, *18*, 4906–4914.

Lund, R. B., Seymour, L., & Kafadar, K. (2001). Temperature trends in the United States. *Environmetrics*, *12*, 673–690.

Mann, M. E., Rutherford, S., Wahl, E., & Ammann, C. (2005). Testing the fidelity of methods used in proxy-based reconstructions of past climate. *Journal of Climate*, *18*, 4097–4107.

Marini, M. M., Olsen, A. R., & Ruben, D. B. (1980). Maximum likelihood estimation in panel studies with missing data. *Sociological Methodology*, *11*, 314–357.

MATLAB Statistics Toolbox (2007). Retrieved August 28, 2007, from http://www.mathworks.com/access/helpdesk/help/toolbox/stats/

Meng, X.-L., & Pedlow, S. (1992). EM: A bibliographic review with missing articles. *Proceedings of the Statistical Computing Section*, *American Statistical Association*. Alexandria, VA: American Statistical Association, 24–27.

Meng, X.-L., & Rubin, D. B. (1991). Using EM to obtain asymptotic variance-covariance matrices: the SEM algorithm. *Journal of the American Statistical Association*, *86*, 899–909.

Orchard, T., & Woodbury, M. A. (1972). A missing information principle: Theory and applications. *Proceedings of the 6th Berkeley Symposium of Mathematical Statistics and Probability*, *1*, 697–715.

Richman, M. B., & Lamb, P. J. (1985). Climatic pattern analysis of three- and seven-day summer rainfall in the central United States: Some methodological considerations and a regionalization. *Journal of Applied Meteorology*, *24*, 1325–1343.

Roth, P. L., Campion, J. E., & Jones, S. D. (1996). The impact of four missing data techniques on validity estimates in human resource management. *Journal of Business and Psychology*, *11*, 101–112.

Rubin, D. B. (1976). Inference and missing data. *Biometrika*, *63*, 581–592.

Rubin, D. B. (1988). An overview of multiple imputation. *Proceedings of the Survey Research Methods Section of the American Statistical Association*, 79–84.

Rutherford, S., Mann, M. E., Osborn, T. J., Bradley, R. S., Briffa, K. R., Hughes, M. K., & Jones, P. D. (2005). Proxy-based Northern hemisphere surface temperature reconstructions: Sensitivity to method, predictor network, target solution and target domain. *Journal of Climate*, *18*, 2308–2329.

Schneider, T. (2001). Analysis of incomplete climate data: Estimation of mean values and covariance matrices and imputation of missing values. *Journal of Climate*, *14*, 853–871.

Spencer, P. L., & Gao, J. (2004). Can gradient information be used to improve variational objective analysis?. *Monthly Weather Review*, *132*, 2977–2994.

Stooksbury, D. E., Idso, C. D., & Hubbard, K. G. (1999). The effects of data gaps on the calculated monthly mean maximum and minimum temperatures in the continental United States: A spatial and temporal study. *Journal of Climate*, *12*, 1524–1533.

Trafalis, T. B., Santosa, B., & Richman, M. B. (2003). Prediction of rainfall from WSR-88D radar using Kernel-based methods. *International Journal of Smart Engineering System Design*, *5*, 429–438.

Vapnik, V. N. (1998). *Statistical learning theory*. New York: Springer.

Wilks, S. S. (1932). Moments and distribution of estimates of population parameters from fragmentary samples. *Annals of Mathematical Statistics*, *3*, 163–195.

Yates, F. (1933). The analysis of replicated experiments when the field results are incomplete. *Empirical Journal of Experimental Agriculture*, *1*, 129–142.

Part II
Applications of AI in Environmental Science

Nonlinear Principal Component Analysis

8

William W. Hsieh

8.1 Introduction

In environmental sciences, one often encounters large datasets with many variables. For instance, one may have a dataset of the monthly sea surface temperature (SST) anomalies ("anomalies" are the departures from the mean) collected at $l = 1{,}000$ grid locations over several decades, i.e. the data are of the form $\mathbf{x} = [x_1, \ldots, x_l]$, where each variable x_i $(i = 1, \ldots, l)$ has n samples. The samples may be collected at times t_k $(k = 1, \ldots, n)$, so each x_i is a time series containing n observations. Since the SST of neighboring grids are correlated, and a dataset with 1,000 variables is quite unwieldy, one looks for ways to condense the large dataset to only a few principal variables. The most common approach is via principal component analysis (PCA), also known as empirical orthogonal function (EOF) analysis (Jolliffe 2002).

In the example with 1,000 variables, imagine we have plotted out all the n samples in the 1,000-dimensional data space, with each sample being a data point in this space. We then try to fit the best straight line through the data points. Mathematically, PCA looks for u, a linear combination of the x_i, and an associated vector $\mathbf{e}$ (which gives the direction of the desired straight line), with

$$u(t) = \mathbf{e} \cdot \mathbf{x}(t)\,, \tag{8.1}$$

so that

$$\langle \|\mathbf{x}(t) - \mathbf{e}\, u(t)\|^2 \rangle \text{ is minimized,} \tag{8.2}$$

where $\langle \cdots \rangle$ denotes a sample mean or time mean. Here u, called the first principal component (PC) (or score), is often a time series, while $\mathbf{e}$, called the first eigenvector (also called an empirical orthogonal function, EOF, or loading), is the first eigenvector of the data covariance matrix $\mathbf{C}$, with elements C_{ij} given by

$$C_{ij} = \frac{1}{n-1} \sum_{k=1}^{n} [x_i(t_k) - \langle x_i \rangle]\, [x_j(t_k) - \langle x_j \rangle]\,. \tag{8.3}$$

Together u and $\mathbf{e}$ make up the first PCA mode. In the above example, $\mathbf{e}$ simply describes a fixed spatial SST anomaly pattern. How strongly this pattern is manifested at a given time is controlled by the time series u.

From the residual, $\mathbf{x} - \mathbf{e}u$, the second PCA mode can similarly be extracted, and so on for the higher modes. In practice, the common algorithms for PCA extract all modes simultaneously (Jolliffe 2002; Preisendorfer 1988). By retaining only the leading modes, PCA has been commonly used to reduce the dimensionality of the dataset, and to extract the main patterns from the dataset.

Principal component analysis (PCA) can be performed using neural network (NN) methods (Oja 1982; Sanger 1989). However, far more interesting is the nonlinear generalization of PCA, where several distinct approaches have been developed (Cherkassky and Mulier 1998). As PCA finds a straight line which passes through the 'middle' of the data cluster, the obvious next step is to generalize the straight line to a curve. The multi-layer perceptron (MLP) model (see Section 1.8) has been adapted to perform nonlinear PCA (Kramer 1991; Hsieh 2004). Alternative approaches are the principal curves method (Hastie and Stuetzle 1989; Hastie et al. 2001), the kernel PCA method (Schölkopf et al. 1998) and the self-organizing

William W. Hsieh (✉)
Department of Earth and Ocean Sciences, 6339 Stores Rd., University of British Columbia, Vancouver, B.C. V6T 1Z4, Canada
Phone: 604-822-2821; fax: 604-822-6088,
email: whsieh@eos.ubc.ca

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

map (SOM) technique (Kohonen 1982; Cherkassky and Mulier 1998).

In this chapter, we examine the use of MLP NN models for nonlinear PCA (NLPCA) in Section 8.2, the overfitting problem associated with NLPCA in Section 8.3, and the extension of NLPCA to closed curve solutions in Section 8.4. MATLAB codes for NLPCA are downloadable from http://www.ocgy.ubc.ca/projects/clim.pred/download.html.The discrete approach by self-organizing maps is presented in Sections 8.5, and the generalization of NLPCA to complex variables in Section 8.6.

8.2 Auto-Associative Neural Networks for NLPCA

The fundamental difference between NLPCA and PCA is that PCA only allows a linear mapping ($u = \mathbf{e} \cdot \mathbf{x}$) between $\mathbf{x}$ and the PC u, while NLPCA allows a nonlinear mapping. To perform NLPCA, Kramer (1991) proposed using the MLP NN in Fig. 8.1a where there are three hidden layers of neurons (i.e. variables) between the input and output layers. The NLPCA is basically a standard MLP NN (see Section 1.8) with four-layers of activation functions (i.e. transfer functions) mapping from the inputs to the outputs. One can view the NLPCA network as composed of two-standard two-layer MLP NNs placed one after the other. The first two-layer network maps from the inputs $\mathbf{x}$ through a hidden layer to the bottleneck layer with only one neuron u, i.e. a nonlinear mapping $u = f(\mathbf{x})$. The next two-layer MLP NN inversely maps from the nonlinear PC (NLPC) u back to the original higher dimensional $\mathbf{x}$-space, with the objective that the outputs $\mathbf{x}' = \mathbf{g}(u)$ be as close as possible to the inputs $\mathbf{x}$, where $\mathbf{g}(u)$ nonlinearly generates a curve in the $\mathbf{x}$-space, hence a 1-dimensional approximation of the original data. Because the target data for the output neurons $\mathbf{x}'$ are simply the input data $\mathbf{x}$, such networks are called auto-associative NNs. To minimize the MSE (mean square error) of this approximation, the objective function (also called cost function or loss function) $J = \langle \|\mathbf{x} - \mathbf{x}'\|^2 \rangle$ is minimized to solve for the parameters of the NN. Squeezing the input information through a bottleneck layer with only one neuron accomplishes the dimensional reduction.

In Fig. 8.1a, the activation function f_1 maps from $\mathbf{x}$, the input column vector of length l, to the first

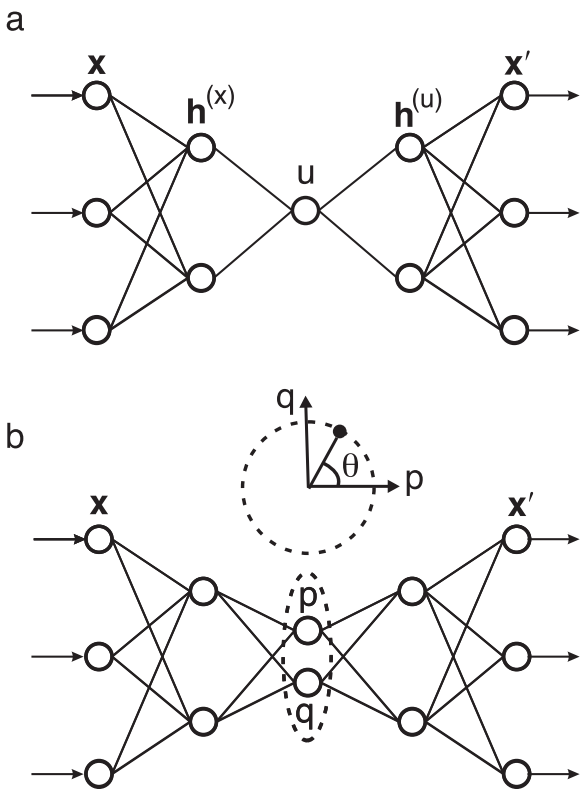


Fig. 8.1 (a) A schematic diagram of the autoassociative feed-forward multi-layer perceptron NN model for performing NLPCA. Between the input layer $\mathbf{x}$ on the left (the 0th layer) and the output layer $\mathbf{x}'$ on the far right (the 4th layer), there are three layers of 'hidden' neurons (the 1st, 2nd and 3rd layers). Layer 2 is the 'bottleneck' with a single neuron u giving the nonlinear principal component (NLPC). Layers 1 and 3, each with m hidden neurons, are called the encoding and decoding layers, respectively. (b) The NN model used for extracting a *closed* curve NLPCA solution. At the bottleneck, there are now two neurons p and q constrained to lie on a unit circle in the p-q plane, giving effectively one free angular variable θ, the NLPC. This network is suited for extracting a closed curve solution (Reprinted from Hsieh 2001. With permission from Blackwell)

hidden layer (the encoding layer), represented by $\mathbf{h}^{(x)}$, a column vector of length m, with elements

$$h_k^{(x)} = f_1((\mathbf{W}^{(x)}\mathbf{x} + \mathbf{b}^{(x)})_k), \tag{8.4}$$

where $\mathbf{W}^{(x)}$ is an $m \times l$ weight matrix, $\mathbf{b}^{(x)}$, a column vector of length m containing the offset (i.e. bias) parameters, and $k = 1, \ldots, m$. Similarly, a second activation function f_2 maps from the encoding layer to the bottleneck layer containing a single neuron, which represents the nonlinear principal component u,

$$u = f_2(\mathbf{w}^{(x)} \cdot \mathbf{h}^{(x)} + \overline{b}^{(x)}). \tag{8.5}$$

The activation function f_1 is generally nonlinear (usually the hyperbolic tangent or the sigmoidal function, though the exact form is not critical), while f_2 is usually taken to be the identity function.

Next, an activation function f_3 maps from u to the third hidden layer (the decoding layer) $\mathbf{h}^{(u)}$,

$$h_k^{(u)} = f_3((\mathbf{w}^{(u)} u + \mathbf{b}^{(u)})_k)\,, \tag{8.6}$$

$(k = 1, \ldots, m)$; followed by f_4 mapping from $\mathbf{h}^{(u)}$ to $\mathbf{x}'$, the output column vector of length l, with

$$x_i' = f_4((\mathbf{W}^{(u)}\mathbf{h}^{(u)} + \overline{\mathbf{b}}^{(u)})_i). \tag{8.7}$$

The objective function $J = \langle \|\mathbf{x} - \mathbf{x}'\|^2 \rangle$ is minimized by finding the optimal values of $\mathbf{W}^{(x)}$, $\mathbf{b}^{(x)}$, $\mathbf{w}^{(x)}$, $\overline{b}^{(x)}$, $\mathbf{w}^{(u)}$, $\mathbf{b}^{(u)}$, $\mathbf{W}^{(u)}$ and $\overline{\mathbf{b}}^{(u)}$. The MSE between the NN output $\mathbf{x}'$ and the original data $\mathbf{x}$ is thus minimized. The NLPCA was implemented using the hyperbolic tangent function for f_1 and f_3, and the identity function for f_2 and f_4, so that

$$u = \mathbf{w}^{(x)} \cdot \mathbf{h}^{(x)} + \overline{b}^{(x)}, \tag{8.8}$$

$$x_i' = (\mathbf{W}^{(u)}\mathbf{h}^{(u)} + \overline{\mathbf{b}}^{(u)})_i. \tag{8.9}$$

Furthermore, we adopt the normalization conditions that $\langle u \rangle = 0$ and $\langle u^2 \rangle = 1$. These conditions are approximately satisfied by modifying the objective function to

$$J = \langle \|\mathbf{x} - \mathbf{x}'\|^2 \rangle + \langle u \rangle^2 + (\langle u^2 \rangle - 1)^2\,. \tag{8.10}$$

The total number of (weight and offset) parameters used by the NLPCA is $2lm + 4m + l + 1$, though the number of effectively free parameters is two less due to the constraints on $\langle u \rangle$ and $\langle u^2 \rangle$.

The choice of m, the number of hidden neurons in both the encoding and decoding layers, follows a general principle of parsimony. A larger m increases the nonlinear modeling capability of the network, but could also lead to overfitted solutions (i.e. wiggly solutions which fit to the noise in the data). If f_4 is the identity function, and $m = 1$, then (8.9) implies that all x_i' are linearly related to a single hidden neuron, hence there can only be a linear relation between the x_i' variables. Thus, for nonlinear solutions, we need to look at $m \geq 2$. Actually, one can use different numbers of neurons in the encoding layer and in the decoding layer; however, keeping them both at m neurons gives roughly the same number of parameters in the forward mapping from $\mathbf{x}$ to u and in the inverse mapping from u to $\mathbf{x}'$. It is also possible to have more than one neuron at the bottleneck layer. For instance, with two bottleneck neurons, the mode extracted will span a 2-D surface instead of a 1-D curve.

Because of local minima in the objective function, there is no guarantee that the optimization algorithm reaches the global minimum. Hence a number of runs with random initial weights and offset parameters was made. Also, a portion (e.g. 15%) of the data was randomly selected as validation data and withheld from the training of the NNs. Runs where the MSE was larger for the validation dataset than for the training dataset were rejected to avoid overfitted solutions. Then the run with the smallest MSE was selected as the solution.

In general, the presence of local minima in the objective function is a major problem for NLPCA. Optimizations started from different initial parameters often converge to different minima, rendering the solution unstable or nonunique. Adding weight penalty terms to the objective function (also called "regularization") is an answer.

The purpose of the weight penalty terms is to limit the nonlinear power of the NLPCA, which came from the nonlinear activation functions in the network. The activation function tanh has the property that given x in the interval $[-L, L]$, one can find a small enough weight w, so that $\tanh(wx) \approx wx$, i.e. the activation function is almost linear. Similarly, one can choose a large enough w, so that tanh approaches a step function, thus yielding Z-shaped solutions. If we can penalize the use of excessive weights, we can limit the degree of nonlinearity in the NLPCA solution. This is achieved with a modified objective function

$$J = \langle \|\mathbf{x} - \mathbf{x}'\|^2 \rangle + \langle u \rangle^2 + (\langle u^2 \rangle - 1)^2 + P \sum_{ki} (W_{ki}^{(x)})^2, \tag{8.11}$$

where P is the weight penalty parameter. A large P increases the concavity of the objective function, and forces the weights $\mathbf{W}^{(x)}$ to be small in magnitude, thereby yielding smoother and less nonlinear solutions than when P is small or zero. Hence, increasing P also reduces the number of effectively free parameters of the model. We have not penalized other weights in the network. In principle, $\mathbf{w}^{(u)}$ also controls the nonlinearity in the inverse mapping from u to $\mathbf{x}'$. However if the nonlinearity in the forward mapping from $\mathbf{x}$ to u is already being limited by penalizing $\mathbf{W}^{(x)}$, then there

is no need to further limit the weights in the inverse mapping.

In summary, one needs to choose m large enough so that the NN model has enough flexibility to approximate the true solution well. The weight penalty P can be regarded as a smoothing parameter, i.e. if P is large enough, zigzags and wiggles in the curve solution can be eliminated. How to choose P and m objectively has only recently been addressed, and is discussed in Section 8.3.

In effect, the linear relation ($u = \mathbf{e} \cdot \mathbf{x}$) in PCA is now generalized to $u = f(\mathbf{x})$, where f can be any nonlinear continuous function representable by an MLP NN mapping from the input layer to the bottleneck layer; and $\langle \|\mathbf{x} - \mathbf{g}(u)\|^2 \rangle$ is minimized. Limitations in the mapping properties of the NLPCA are discussed by Newbigging et al. (2003). The residual, $\mathbf{x} - \mathbf{g}(u)$, can be input into the same network to extract the second NLPCA mode, and so on for the higher modes.

That the classical PCA is indeed a linear version of this NLPCA can be readily seen by replacing all the activation functions with the identity function, thereby removing the nonlinear modeling capability of the NLPCA. Then the forward map to u involves only a linear combination of the original variables as in the PCA.

In the classical linear approach, there is a well-known dichotomy between PCA and rotated PCA (RPCA) (Richman 1986). In PCA, the linear mode which accounts for the most variance of the dataset is sought. However, as illustrated in Preisendorfer (1988, Fig. 7.3), the resulting eigenvectors may not align close to local data clusters, so the eigenvectors may not represent actual physical states well. One application of RPCA methods is to rotate the PCA eigenvectors, so they point closer to the local clusters of data points (Preisendorfer 1988). Thus the rotated eigenvectors may bear greater resemblance to actual physical states (though they account for less variance) than the unrotated eigenvectors, hence RPCA is also widely used (Richman 1986; von Storch and Zwiers 1999). As there are many possible criteria for rotation, there are many RPCA schemes, among which the varimax (Kaiser 1958) scheme is perhaps the most popular. We will compare NLPCA with PCA and RPCA in the following subsection.

8.2.1 Applications of NLPCA

The NLPCA has been applied to the Lorenz (1963) three-component chaotic system (Monahan 2000; Hsieh 2001). For the tropical Pacific climate variability, the NLPCA has been used to study the SST field (Monahan 2001; Hsieh 2001) and the sea level pressure (SLP) field (Monahan 2001). The Northern Hemisphere atmospheric variability (Monahan et al. 2000, 2001) and the subsurface thermal structure of the Pacific Ocean (Tang and Hsieh 2003) have also been investigated by the NLPCA. In remote sensing, Del Frate and Schiavon (1999) applied NLPCA to the inversion of radiometric data to retrieve atmospheric profiles of temperature and water vapour.

The tropical Pacific climate system contains the famous interannual variability known as the El Niño-Southern Oscillation (ENSO), a coupled atmosphere-ocean interaction involving the oceanic phenomenon El Niño and the associated atmospheric phenomenon, the Southern Oscillation. The coupled interaction results in anomalously warm SST in the eastern equatorial Pacific during El Niño episodes, and cool SST in the central equatorial Pacific during La Niña episodes (Philander 1990; Diaz and Markgraf 2000). ENSO is an irregular oscillation, but spectral analysis does reveal a broad spectral peak at the 4–5 year period. Hsieh (2001) used the tropical Pacific SST data (1950–1999) to make a three-way comparison between NLPCA, RPCA and PCA. The tropical Pacific SST anomaly (SSTA) data (i.e. the SST data with the climatological seasonal cycle removed) were pre-filtered by PCA, with only the three leading modes retained. PCA modes 1, 2 and 3 accounted for 51.4%, 10.1% and 7.2%, respectively, of the variance in the SSTA data. Due to the large number of spatially gridded variables, NLPCA could not be applied directly to the SSTA time series, as this would lead to a huge NN with the number of model parameters vastly exceeding the number of samples. Instead, the first three PCs (PC1, PC2 and PC3) were used as the input $\mathbf{x}$ for the NLPCA network.

The data are shown as dots in a scatter plot in the PC1-PC2 plane (Fig. 8.2), where the cool La Niña states lie in the upper left corner, and the warm El Niño states in the upper right corner. The NLPCA solution is a U-shaped curve linking the La Niña states at one end (low u) to the El Niño states at the other end (high u),

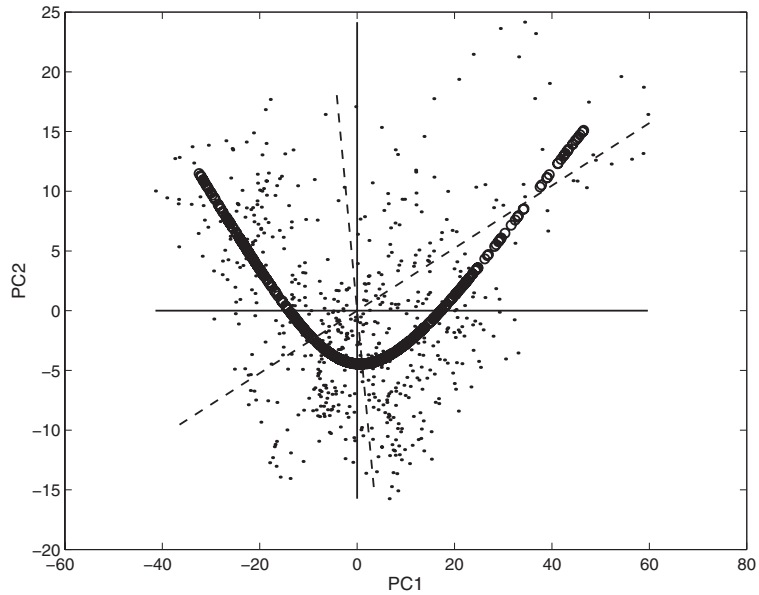


Fig. 8.2 Scatter plot of the SST anomaly (SSTA) data (shown as dots) in the PC1-PC2 plane, with the El Niño states lying in the upper right corner, and the La Niña states in the upper left corner. The PC2 axis is stretched relative to the PC1 axis for better visualization. The first mode NLPCA approximation to the data is shown by the (overlapping) small circles, which traced out a U-shaped curve. The first PCA eigenvector is oriented along the horizontal line, and the second PCA, by the vertical line. The varimax method rotates the two PCA eigenvectors in a counterclockwise direction, as the rotated PCA (RPCA) eigenvectors are oriented along the dashed lines. (As the varimax method generates an orthogonal rotation, the angle between the two RPCA eigenvectors is 90° in the 3-dimensional PC1-PC2-PC3 space) (Reprinted from Hsieh 2001. With permission from Blackwell)

similar to that found originally by Monahan (2001). In contrast, the first PCA eigenvector lies along the horizontal line, and the second PCA, along the vertical line (Fig. 8.2). It is easy to see that the first PCA eigenvector describes a somewhat unphysical oscillation, as there are no dots (data) close to either ends of the horizontal line. For the second PCA eigenvector, there are dots close to the bottom of the vertical line, but no dots near the top end of the line, i.e. one phase of the mode 2 oscillation is realistic, but the opposite phase is not. Thus if the underlying data has a nonlinear structure but we are restricted to finding linear solutions using PCA, the energy of the nonlinear oscillation is scattered into multiple PCA modes, many of which represent unphysical linear oscillations.

For comparison, a varimax rotation (Kaiser 1958; Preisendorfer 1988), was applied to the first three PCA eigenvectors. The varimax criterion can be applied to either the loadings or the PCs depending on one's objectives (Richman 1986; Preisendorfer 1988); here it is applied to the PCs. The resulting first RPCA eigenvector, shown as a dashed line in Fig. 8.2, spears through the cluster of El Niño states in the upper right corner, thereby yielding a more accurate description of the El Niño anomalies (Fig. 8.3c) than the first PCA mode (Fig. 8.3a), which did not fully represent the intense warming of Peruvian waters. The second RPCA eigenvector, also shown as a dashed line in Fig. 8.2, did not improve much on the second PCA mode, with the PCA spatial pattern shown in Fig. 8.3b, and the RPCA pattern in Fig. 8.3d). In terms of variance explained, the first NLPCA mode explained 56.6% of the variance, versus 51.4% by the first PCA mode, and 47.2% by the first RPCA mode.

With the NLPCA, for a given value of the NLPC u, one can map from u to the three PCs. This is done by assigning the value u to the bottleneck neuron and mapping forward using the second half of the network in Fig. 8.1a. Each of the three PCs can be multiplied by its associated PCA (spatial) eigenvector, and the three added together to yield the spatial pattern

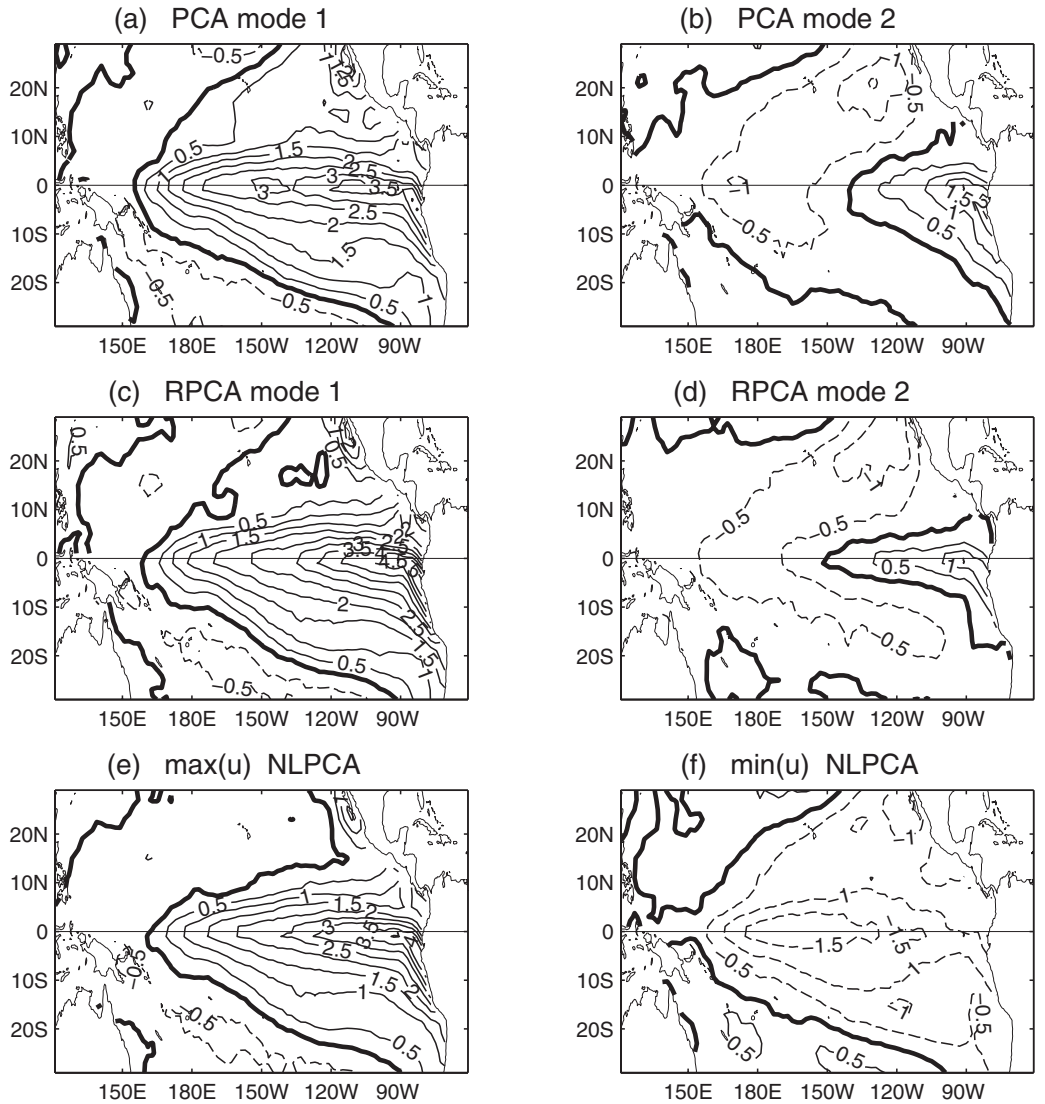


Fig. 8.3 The SSTA patterns (in °C) of the PCA, RPCA and the NLPCA. The first and second PCA spatial modes are shown in (a) and (b) respectively, (both with their corresponding PCs at maximum value). The first and second varimax RPCA spatial modes are shown in (c) and (d) respectively, (both with their corresponding RPCs at maximum value). The anomaly pattern as the NLPC u of the first NLPCA mode varies from (e) maximum (strong El Niño) to (f) its minimum (strong La Niña). With a contour interval of 0.5°C , the positive contours are shown as solid curves, negative contours, dashed curves, and the zero contour, a thick curve (Reprinted from Hsieh 2004. With permission from American Geophysical Union)

for that particular value of u. Unlike PCA which gives the same spatial anomaly pattern except for changes in the amplitude as the PC varies, the NLPCA spatial pattern generally varies continuously as the NLPC changes. Figure 8.3e, f show respectively the spatial anomaly patterns when u has its maximum value (corresponding to the strongest El Niño) and when u has its minimum value (strongest La Niña). Clearly the asymmetry between El Niño and La Niña, i.e. the cool anomalies during La Niña episodes (Fig. 8.3f) are observed to center much further west than the warm anomalies during El Niño (Fig. 8.3e) (Hoerling et al. 1997), is well captured by the first NLPCA mode – in contrast, the PCA mode 1 gives a La Niña which is simply the mirror image of the El Niño (Fig. 8.3a). The asymmetry explains why El

Niño has been known by Peruvian fishermen for many centuries due to its strong SSTA off the coast of Peru and its devastation of the Peruvian fishery, whereas the La Niña, with its weak manifestation in the Peruvian waters, was not appreciated until the last two decades of the 20th century.

In summary, PCA is used for two main purposes: (i) to reduce the dimensionality of the dataset, and (ii) to extract features or recognize patterns from the dataset. It is primarily purpose (ii) where PCA can be improved upon. Both RPCA and NLPCA take the PCs from PCA as input. However, instead of multiplying the PCs by a fixed orthonormal rotational matrix, as performed in the varimax RPCA approach, NLPCA performs a nonlinear mapping of the PCs. RPCA sacrifices on the amount of variance explained, but by rotating the PCA eigenvectors, RPCA eigenvectors tend to point more towards local data clusters and are therefore more representative of physical states than the PCA eigenvectors.

With a linear approach, it is generally impossible to have a solution simultaneously (a) explaining maximum global variance of the dataset and (b) approaching local data clusters, hence the dichotomy between PCA and RPCA, with PCA aiming for (a) and RPCA for (b). Hsieh (2001) pointed out that with the more flexible NLPCA method, both objectives (a) and (b) may be attained together, thus the nonlinearity in NLPCA unifies the PCA and RPCA approaches. It is easy to see why the dichotomy between PCA and RPCA in the linear approach automatically vanishes in the nonlinear approach. By increasing m, the number of hidden neurons in the encoding layer (and the decoding layer), the solution is capable of going through all local data clusters while maximizing the global variance explained. (In fact, for large enough m, NLPCA can pass through all data points, though this will in general give an undesirable, overfitted solution.)

The tropical Pacific SST example illustrates that with a complicated oscillation like the El Niño-La Niña phenomenon, using a linear method such as PCA results in the nonlinear mode being scattered into several linear modes (in fact, all three leading PCA modes are related to this phenomenon) – hence the importance of the NLPCA as a unifier of the separate linear modes. In the study of climate variability, the wide use of PCA methods has created the somewhat misleading view that our climate is dominated by a number of spatially fixed oscillatory patterns, which may in fact be due to the limitation of the linear method. Applying NLPCA to the tropical Pacific SSTA, we found no spatially fixed oscillatory patterns, but an oscillation evolving in space as well as in time.

8.3 Overfitting in NLPCA

When using nonlinear machine learning methods, the presence of noise in the data can lead to overfitting. When plentiful data are available (i.e. far more samples than model parameters), overfitting is not a problem when performing nonlinear regression on noisy data. Unfortunately, even with plentiful data, overfitting is a problem when applying NLPCA to noisy data (Hsieh 2001; Christiansen 2005; Hsieh 2007). As illustrated in Fig. 8.4, overfitting in NLPCA can arise from the geometry of the problem, rather than from the scarcity of data. Here for a Gaussian-distributed data cloud, a nonlinear model with enough flexibility will find the zigzag solution of Fig. 8.4b as having a smaller MSE than the linear solution in Fig. 8.4a. Since the distance between the point A and a, its projection on the NLPCA curve, is smaller in Fig. 8.4b than the corresponding distance in Fig. 8.4a, it is easy to see that the more zigzags there are in the curve, the smaller is the MSE. However, the two neighboring points A and B, on opposite sides of an ambiguity line, are projected far apart on the NLPCA curve in Fig. 8.4b. Thus simply searching for the solution which gives the smallest MSE does not guarantee that NLPCA will find a good solution in a highly noisy dataset.

Hsieh (2001) added weight penalty to the Kramer (1991) NLPCA model to smooth out excessively wiggly solutions, but did not provide an objective way to select the optimal weight penalty parameter P. With NLPCA, if the overfitting arise from the data geometry (as in Fig. 8.4b) and not from data scarcity, using independent data to validate the MSE from the various models is not a viable method for choosing the appropriate P. Instead, Hsieh (2007) proposed an "inconsistency" index for detecting the projection of neighboring points to distant parts of the NLPCA curve, and use the index to choose the appropriate P.

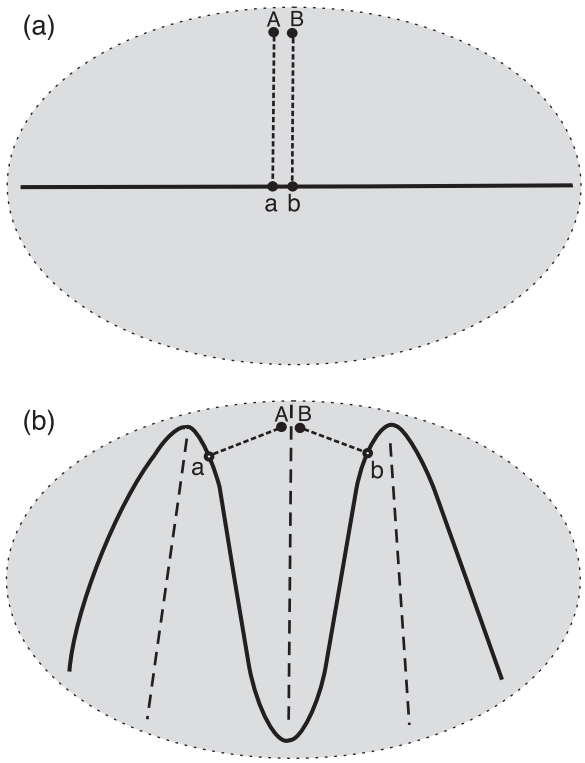


Fig. 8.4 Schematic diagram illustrating overfitting on noisy data. (a) PCA solution for a Gaussian data cloud, with two neighboring points A and B shown projecting to the points a and b on the PCA straight line solution. (b) A zigzag NLPCA solution found by a flexible enough nonlinear model. Dashed lines illustrate ambiguity lines where neighboring points (e.g. A and B) on opposite sides of these lines are projected to a and b, far apart on the NLPCA curve (Reprinted from Hsieh 2007. With permission from Elsevier)

For each data point $\mathbf{x}$, find its nearest neighbor $\tilde{\mathbf{x}}$. The NLPC for $\mathbf{x}$ and $\tilde{\mathbf{x}}$ are u and $\tilde{u}$, respectively. With $C(u, \tilde{u})$ denoting the (Pearson) correlation between all the pairs $(u, \tilde{u})$, the inconsistency index I was defined in Hsieh (2007) as

$$I = 1 - C(u, \tilde{u}). \tag{8.12}$$

If for some nearest neighbor pairs, u and $\tilde{u}$ are assigned very different values, $C(u, \tilde{u})$ would have a lower value, leading to a larger I, indicating greater inconsistency in the NLPC mapping. With u and $\tilde{u}$ standardized to having zero mean and unit standard deviation, (8.12) is equivalent to

$$I = \frac{1}{2} \langle (u - \tilde{u})^2 \rangle . \tag{8.13}$$

In statistics, various criteria, often in the context of linear models, have been developed to select the right amount of model complexity so neither overfitting nor underfitting occurs. These criteria are often called "information criteria" (IC) (von Storch and Zwiers 1999). An IC is typically of the form

$$\text{IC} = \text{MSE} + \text{complexity term}, \tag{8.14}$$

where MSE is evaluated over the training data and the complexity term is larger when a model has more free parameters. The IC is evaluated over a number of models with different free parameters, and the model with the minimum IC is selected as the best. As the presence of the complexity term in the IC penalizes models which use excessive number of free parameters to attain low MSE, choosing the model with the minimum IC would rule out complex models with overfitted solutions.

In Hsieh (2007), the data were randomly divided into a training data set and a validation set (containing 85% and 15% of the original data, respectively), and for every given value of P and m, the model was trained a number of times from random initial weights, and model runs where the MSE evaluated over the validation data was larger than the MSE over the training data were discarded. To choose among the model runs which had passed the validation test, a new holistic IC to deal with the type of overfitting arising from the broad data geometry (Fig. 8.4b) was introduced as

$$H = \text{MSE} + \text{inconsistency term} \tag{8.15}$$

$$= \text{MSE} - C(u, \tilde{u}) \times \text{MSE} \; = \; \text{MSE} \times I, \tag{8.16}$$

where MSE and C were evaluated over all (training and validation) data, inconsistency was penalized, and the model run with the smallest H value was selected as the best. As the inconsistency term only prevents overfitting arising from the broad data geometry, validation data are still needed to prevent "local" overfitting from excessive number of model parameters, since H, unlike (8.14), does not contain a complexity term.

Consider the test problem in Hsieh (2007): For a random number t uniformly distributed in the interval $(-1, 1)$, the signal $\mathbf{x}^{(s)}$ was generated by using a quadratic relation

$$x_1^{(s)} = t, \qquad x_2^{(s)} = \frac{1}{2} t^2. \tag{8.17}$$

Isotropic Gaussian noise was added to the signal $\mathbf{x}^{(s)}$ to give the noisy data $\mathbf{x}$ with 500 samples. NLPCA was performed on the data using the network in Fig. 8.1a with $m = 4$ and with the weight penalty P at various values (10, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 0). For each value of P, the model training was done 30 times starting from random initial weights, and model runs where the MSE evaluated over the validation data was larger than the MSE over the training data were deemed ineligible. In the traditional approach, among the eligible runs over the range of P values, the one with the lowest MSE over all (training and validation) data was selected as the best. Figure 8.5a shows this solution where the zigzag curve retrieved by NLPCA is very different from the theoretical parabolic signal (8.17), demonstrating the pitfall of selecting the lowest MSE run.

In contrast, in Fig. 8.5b, among the eligible runs over the range of P values, the one with the lowest information criterion H was selected. This solution, which has a much larger weight penalty ($P = 0.1$) than that in Fig. 8.5a ($P = 10^{-4}$), shows less wiggly behaviour and better agreement with the theoretical parabolic signal.

Even less wiggly solutions can be obtained by changing the error norm used in the objective function from the mean square error to the mean absolute error (MAE), i.e. replacing $\langle \|\mathbf{x} - \mathbf{x}'\|^2 \rangle$ by $\langle \sum_j |x_j - x'_j| \rangle$ in equation (8.11). The MAE norm is known to be robust to outliers in the data (Bishop 1995, p. 210). Figure 8.5c is the solution selected based on minimum H with the MAE norm used. While wiggles are eliminated, the solution underestimates the curvature in the parabolic signal. The rest of this paper uses the MSE norm.

In summary, with noisy data, not having plentiful samples could cause a flexible nonlinear model to overfit. In the limit of infinite samples, overfitting cannot occur in nonlinear regression, but can still occur in NLPCA due to the geometric shape of the data distribution. A new inconsistency index I for detecting the projection of neighboring points to distant parts of the NLPCA curve has been introduced, and incorporated into a holistic IC H to select the model with the appropriate weight penalty parameter and the appropriate number of hidden neurons. An alternative approach for model selection was proposed by Webb (1999), who applied a constraint on the Jacobian in the objective function.

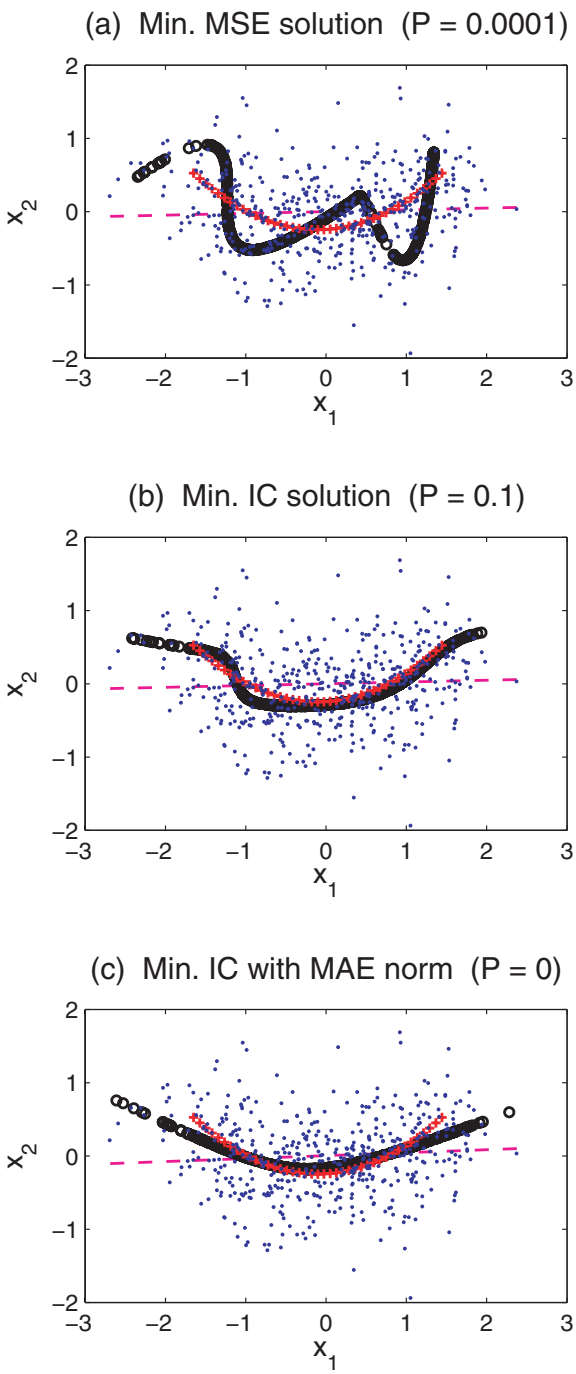


Fig. 8.5 The NLPCA solution (shown as densely overlapping black circles) for the synthetic dataset (dots), with the parabolic signal curve indicated by "+" and the linear PCA solution by the dashed line. The solution was selected from the multiple runs over a range of P values based on (a) minimum MSE, (b) minimum IC H, and (c) minimum IC together with the MAE norm (Reprinted from Hsieh 2007. With permission from Elsevier)

8.4 NLPCA for Closed Curves

While the NLPCA is capable of finding a continuous open curve solution, there are many phenomena involving waves or quasi-periodic fluctuations, which call for a continuous closed curve solution. Kirby and Miranda (1996) introduced an NLPCA with a circular node at the network bottleneck [henceforth referred to as the NLPCA(cir)], so that the nonlinear principal component (NLPC) as represented by the circular node is an angular variable θ, and the NLPCA(cir) is capable of approximating the data by a closed continuous curve. Figure 8.1b shows the NLPCA(cir) network, which is almost identical to the NLPCA of Fig. 8.1a, except at the bottleneck, where there are now two neurons p and q constrained to lie on a unit circle in the p-q plane, so there is only one free angular variable θ, the NLPC.

At the bottleneck in Fig. 8.1b, analogous to u in (8.8), we calculate the pre-states p_o and q_o by

$$p_o = \mathbf{w}^{(x)} \cdot \mathbf{h}^{(x)} + \overline{b}^{(x)}, \quad \text{and}$$
$$q_o = \tilde{\mathbf{w}}^{(x)} \cdot \mathbf{h}^{(x)} + \tilde{b}^{(x)}, \tag{8.18}$$

where $\mathbf{w}^{(x)}$, $\tilde{\mathbf{w}}^{(x)}$ are weight parameter vectors, and $\overline{b}^{(x)}$ and $\tilde{b}^{(x)}$ are offset parameters. Let

$$r = (p_o^2 + q_o^2)^{1/2}, \tag{8.19}$$

then the circular node is defined with

$$p = p_o/r, \quad \text{and} \quad q = q_o/r, \tag{8.20}$$

satisfying the unit circle equation $p^2 + q^2 = 1$. Thus, even though there are two variables p and q at the bottleneck, there is only one angular degree of freedom from θ (Fig. 8.1b), due to the circle constraint. The mapping from the bottleneck to the output proceeds as before, with (8.6) replaced by

$$h_k^{(u)} = f_3((\mathbf{w}^{(u)} p + \tilde{\mathbf{w}}^{(u)} q + \mathbf{b}^{(u)})_k). \tag{8.21}$$

When implementing NLPCA(cir), Hsieh (2001) found that there are actually two possible configurations: (i) A restricted configuration where the constraints $\langle p \rangle = 0 = \langle q \rangle$ are applied, and (ii) a general configuration without the constraints. With (i), the constraints can be satisfied approximately by adding the extra terms $\langle p \rangle^2$ and $\langle q \rangle^2$ to the objective function. If a closed curve solution is sought, then (i) is better than (ii) as it has effectively two fewer parameters. However, (ii), being more general than (i), can more readily model open curve solutions like a regular NLPCA. The reason is that if the input data mapped onto the p-q plane covers only a segment of the unit circle instead of the whole circle, then the inverse mapping from the p-q space to the output space will yield a solution resembling an open curve. Hence, given a dataset, (ii) may yield either a closed curve or an open curve solution. It uses $2lm + 6m + l + 2$ parameters.

Hsieh (2007) found that the IC H not only alleviates overfitting in open curve solution, but also chooses between open and closed curve solutions when using NLPCA(cir) in configuration (ii). The inconsistency index and the IC are now obtained from

$$I = 1 - \frac{1}{2}\left[C(p, \tilde{p}) + C(q, \tilde{q})\right], \quad \text{and}$$
$$H = \text{MSE} \times I, \tag{8.22}$$

where p and q are from the bottleneck (Fig. 8.1b), and $\tilde{p}$ and $\tilde{q}$ are the corresponding nearest neighbor values.

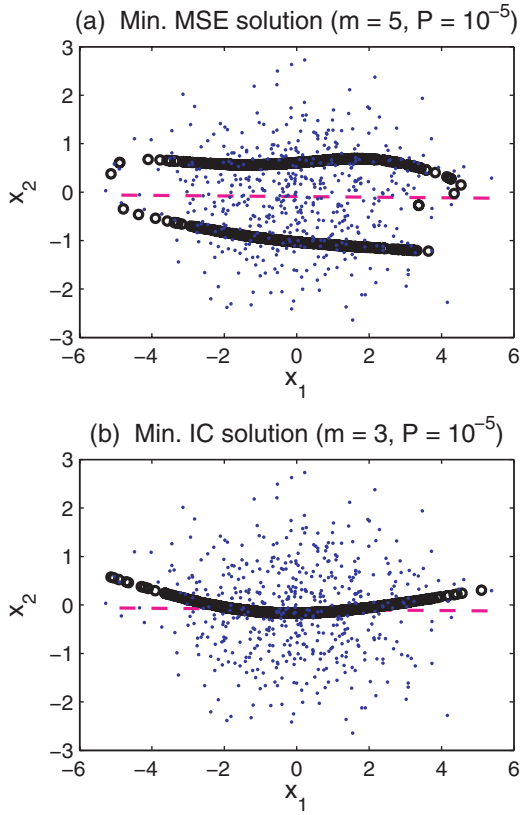


Fig. 8.6 The NLPCA(cir) mode 1 for a Gaussian dataset, with the solution selected based on (a) minimum MSE and (b) minimum IC. The PCA mode 1 solution is shown as a dashed line (Reprinted from Hsieh 2007. With permission from Elsevier)

For a test problem, consider a Gaussian data cloud (with 500 samples) in 2-dimensional space, where the standard deviation along the x_1 axis was double that along the x_2 axis. The dataset was analyzed by the NLPCA(cir) model with $m = 2, \ldots, 5$ and $P = 10$, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 0. From all the runs, the solution selected based on the minimum MSE has $m = 5$ (and $P = 10^{-5}$) (Fig. 8.6a), while that selected based on minimum H has $m = 3$ (and $P = 10^{-5}$) (Fig. 8.6b). The minimum MSE solution has (normalized) MSE = 0.370, $I = 9.50$ and $H = 3.52$, whereas the minimum H solution has the corresponding values of 0.994, 0.839 and 0.833, respectively, where for easy comparison with the linear mode, these values for the nonlinear solutions have been normalized upon division by the corresponding values from the linear PCA mode 1. Thus the IC correctly selected a nonlinear solution (Fig. 8.6b) which is similar to the linear solution. It also rejected the closed curve solution of Fig. 8.6a, in favour of the open curve solution of Fig. 8.6b, despite its much larger MSE.

For an application of NLPCA(cir) on real data, consider the quasi-biennial oscillation (QBO), which dominates over the annual cycle or other variations in the equatorial stratosphere, with the period of oscillation varying roughly between 22 and 32 months. Average zonal (i.e. westerly) winds at 70, 50, 40, 30, 20, 15 and 10 hPa (i.e. from about 20 to 30 km altitude) during 1956–2006 were studied. After the 51-year means were removed, the zonal wind anomalies U at seven vertical levels in the stratosphere became the seven inputs to the NLPCA(cir) network (Hamilton and Hsieh 2002; Hsieh 2007). Since the data were not very noisy (Fig. 8.7), a rather complex model was used, with m ranging from 5 to 9, and $P = 10^{-1}$, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 0. The smallest H occurred when

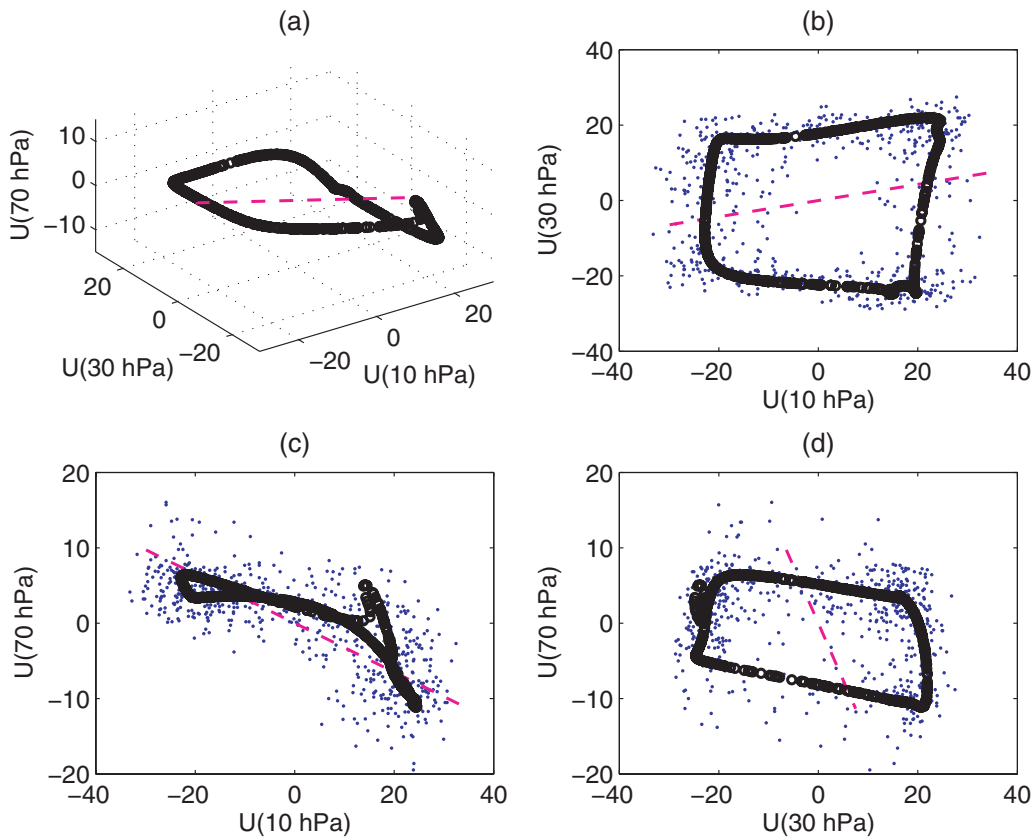


Fig. 8.7 The NLPCA(cir) mode 1 solution for the equatorial stratospheric zonal wind anomalies. For comparison, the PCA mode 1 solution is shown by the dashed line. Only three out of seven dimensions are shown, namely the zonal velocity anomaly U at the top, middle and bottom levels (10, 30 and 70 hPa). Panel (a) gives a 3-D view, while (b–d) give 2-D views (Reprinted from Hsieh 2007. With permission from Elsevier)

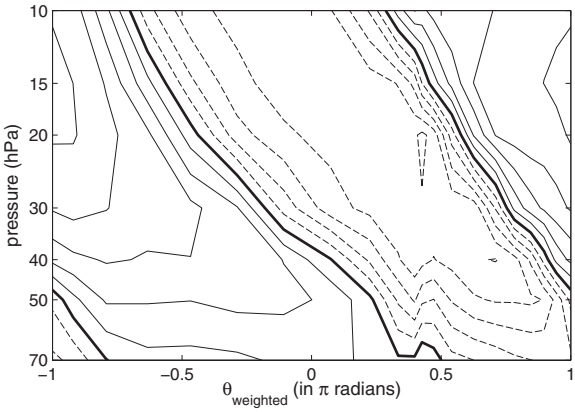


Fig. 8.8 Contour plot of the NLPCA(cir) mode 1 zonal wind anomalies as a function of pressure and phase θ_{weighted}, where θ_{weighted} is θ weighted by the histogram distribution of θ (see Hamilton and Hsieh 2002). Thus θ_{weighted} is more representative of actual time during a cycle than θ. Contour interval is 5 m s^{-1}, with westerly winds indicated by solid lines, easterlies by dashed lines, and zero contours by thick lines (Reprinted from Hsieh 2007. With permission from Elsevier)

$m = 8$ and $P = 10^{-5}$, with the closed curve solution shown in Fig. 8.7. Thus in this example, by choosing a rather large m and a small P, the H IC justified having considerable model complexity, including the wiggly behaviour seen in the 70 hPa wind (Fig. 8.7c). The wiggly behaviour can be understood by viewing the phase-pressure contour plot of the zonal wind anomalies (Fig. 8.8): As the easterly wind anomaly descends with time (i.e. as phase increases), wavy behaviour is seen in the 40, 50 and 70 hPa levels at θ_{weighted} around 0.4–0.5. This example demonstrates the benefit of having an IC to objectively decide on how smooth or wiggly the fitted curve should be.

The observed strong asymmetries between the easterly and westerly phases of the QBO (Hamilton 1998; Baldwin et al. 2001) are captured by this NLPCA(cir) mode – e.g. the much more rapid transition from easterlies to westerlies than the reverse transition, and the much deeper descend of the easterlies than the westerlies (Fig. 8.8). For comparison, Hamilton and Hsieh (2002) constructed a *linear* model of θ, which was unable to capture the observed strong asymmetry between easterlies and westerlies.

The actual time series of the wind measured at a particular height level is somewhat noisy and it is often desirable to have a smoother representation of the QBO time series which captures the essential features at all vertical levels. Also, the reversal of the wind from westerly to easterly and vice versa occurs at different times for different height levels, rendering it difficult to define the phase of the QBO. Hamilton and Hsieh (2002) found that the phase of the QBO as defined by the NLPC θ is more accurate than previous attempts to characterize the phase, leading to a stronger link between the QBO and northern hemisphere polar stratospheric temperatures in winter (the Holton-Tan effect) (Holton and Tan 1980) than previously found.

The NLPCA(cir) approach has also been used successfully in capturing the non-sinusoidal propagation of underwater sandbars off beaches in the Netherlands and Japan (Ruessink et al. 2004). Hsieh and Wu (2002) developed a nonlinear singular spectrum analysis method based on the NLPCA(cir) model.

8.5 Self-Organizing Maps

In this section, we examine a discrete version of NLPCA. The goal of *clustering* or cluster analysis is to group the data into a number of subsets or "clusters", such that the data within a cluster are more closely related to each other than data from other clusters. By projecting all data belonging to a cluster to the cluster center, data compression can be achieved.

The *self-organizing map* (SOM) method, introduced by Kohonen (1982, 2001), approximates a dataset in multidimensional space by a flexible grid (typically of one or two dimensions) of cluster centers. Widely used for clustering, SOM can also be regarded as a discrete version of NLPCA (Cherkassky and Mulier 1998).

As with many neural network models, self-organizing maps have a biological background (Rojas 1996). In neurobiology, it is known that many structures in the cortex of the brain are 2-D or 1-D. In contrast, even the perception of color involves three types of light receptors. Besides color, human vision also processes information about the shape, size, texture, position and movement of an object. So the question naturally arises on how 2-D networks of neurons in the brain can process higher dimensional signals.

Among various possible grids, the rectangular grid is most commonly used by SOM. For a 2-dimensional rectangular grid, the grid points $\mathbf{i}_j = (l, m)$, where l and m take on integer values, i.e. $l = 1, \ldots,$

L, $m = 1, \ldots, M$, and $j = 1, \ldots, LM$. (If a 1-dimensional grid is desired, simply set $M = 1$).

To initialize the training process, PCA is usually first performed on the dataset, and a grid $\mathbf{z}_j(0)$ is placed on the plane spanned by the two leading PCA eigenvectors, with each $\mathbf{z}_j(0)$ corresponding to $\mathbf{i}_j$ on the integer grid. As training proceeded, the initial flat 2D surface of $\mathbf{z}_j(0)$ is bended to fit the data. The original SOM was trained in a flow-through manner (i.e. samples are presented one at a time during training), though algorithms for batch training are now also available. In flow-through training, there are two steps to be iterated, starting with $n = 1$:

Step (i): At the nth iteration, select a sample $\mathbf{x}(n)$ from the data space, and find among the points $\mathbf{z}_j(n-1)$, the one with the closest (Euclidean) distance to $\mathbf{x}(n)$. Call this closest neighbor $\mathbf{z}_k(n)$, corresponding to the integer grid point $\mathbf{i}_k(n)$.

Step (ii): Let

$$\mathbf{z}_j(n) = \mathbf{z}_j(n-1) + \eta\, h(\|\mathbf{i}_j - \mathbf{i}_k(n)\|^2) \times [\mathbf{x}(n) - \mathbf{z}_j(n-1)], \tag{8.23}$$

where η is the learning rate parameter and h is a neighborhood or kernel function. The neighborhood function gives more weight to the grid points $\mathbf{i}_j$ near $\mathbf{i}_k(n)$, than those far away, an example being a Gaussian drop-off with distance. Note the distance of the neighbors are computed for the fixed grid points ($\mathbf{i}_j = (l, m)$), not for their corresponding positions $\mathbf{z}_j(n)$ in the data space. Typically, as n increases, the learning rate η is decreased gradually from the initial value of 1 towards 0, while the width of the neighborhood function is also gradually narrowed (Cherkassky and Mulier 1998).

As an example, consider the famous Lorenz 'butterfly'-shaped attractor from chaos theory (Lorenz 1963). Describing idealized atmospheric convection, the Lorenz system is governed by three (nondimensionalized) differential equations:

$$\dot{x} = -ax + ay, \quad \dot{y} = -xz + bx - y, \quad \dot{z} = xy - cz, \tag{8.24}$$

where the overhead dot denotes a time derivative, and a, b and c are three parameters. A chaotic system is generated by choosing $a = 10, b = 28$, and $c = 8/3$. The Lorenz data is fitted by a 2-dimensional SOM (from the MATLAB neural network toolbox) in Fig. 8.9, and by a 1-dimensional SOM in Fig. 8.10. The

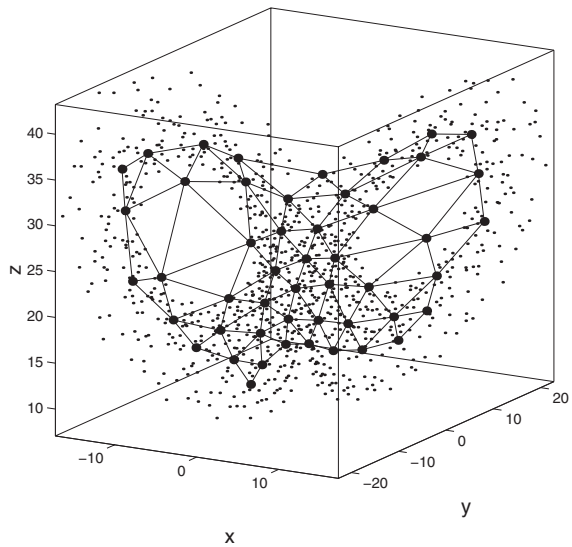


Fig. 8.9 A 2-dimensional self-organizing map (SOM) where a 7×7 mesh is fitted to the Lorenz (1963) attractor data

1-dimensional fit resembles a discrete version of the NLPCA solution found using auto-associative neural networks (Monahan 2000).

SOM has been applied to the classification of satellite-sensed ocean color (Yacoub et al. 2001), sea surface temperature (Richardson et al. 2003), sea level height (Hardman-Mountford et al. 2003), scatterometer winds (Richardson et al. 2003) and ocean currents

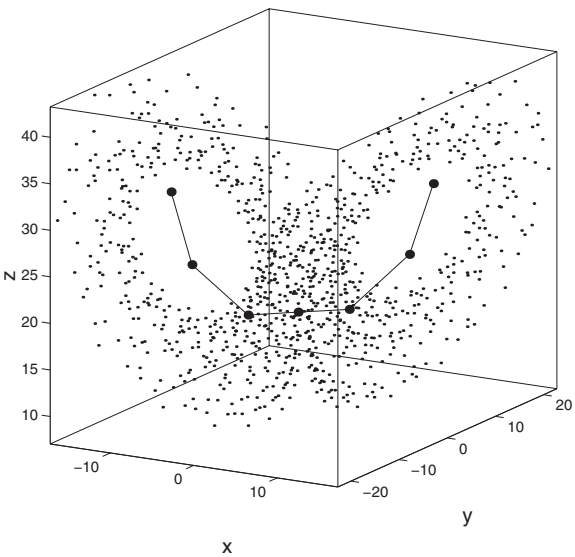


Fig. 8.10 A 1-dimensional self-organizing map (SOM) where a curve with seven nodes is fitted to the Lorenz (1963) attractor data

(Liu et al. 2006). Villman et al. (2003) applied SOM to not only clustering low-dimensional spectral data from the LANDSAT thematic mapper, but also to the high-dimensional hyperspectral AVIRIS (Airbourn Visible-Near Infrared Imaging Spectrometer) data where there are about 200 frequency bands. A 2-D SOM with a mesh of 40×40 was applied to AVIRIS data to classify the geology of the land surface.

Cavazos (1999) applied a 2×2 SOM to cluster the winter daily precipitation over 20 grid points in northeastern Mexico and southeastern Texas. From the wettest and driest clusters, composites of the 500 hPa geopotential heights and sea level pressure were generated, yielding the large scale meteorological conditions associated with the wettest and driest clusters. Cavazos (2000) and Cavazos et al. (2002) also applied SOMs with more clusters to other areas of the world.

8.6 NLPCA for Complex Variables

Complex principal component analysis (CPCA) is PCA applied to complex variables. In the first type of application, a 2-dimensional vector such as the wind (u, v) can be treated as a complex variable $w = u + \mathrm{i}v$ and analyzed by CPCA. In the second type of application, a real time-varying field can be complexified by the Hilbert transform and analyzed by CPCA, often called Hilbert PCA (von Storch and Zwiers 1999) to distinguish from the first type of application.

Earlier in this chapter, we have examined the auto-associative multi-layer perceptron NN approach of Kramer (1991) for performing nonlinear PCA. Here we will discuss how the same approach can be applied to complex variables, giving rise to nonlinear complex PCA (NLCPCA).

In the real domain, a common nonlinear activation function is the hyperbolic tangent function $\tanh(x)$, bounded between -1 and $+1$ and analytic everywhere. For a complex activation function to be bounded and analytic everywhere, it has to be a constant function (Clarke 1990), as Liouville's theorem states that entire functions (i.e. functions that are analytic on the whole complex plane) which are bounded are always constants. The function $\tanh(z)$ in the complex domain has an infinite number of singularities located at $(\frac{1}{2} + l)\,\pi i$, $l \in \mathbb{N}$. Using functions like $\tanh(z)$ (without any constraint) leads to non-convergent solutions (Nitta 1997).

Traditionally, the complex activation functions used focussed mainly on overcoming the unbounded nature of the analytic functions in the complex domain. Some complex activation functions basically scaled the magnitude (amplitude) of the complex signals but preserved their arguments (phases) (Georgiou and Koutsougeras 1992; Hirose 1992), hence they are less effective in learning non-linear variations in the argument. A more traditional approach has been to use a "split" complex nonlinear activation function (Nitta 1997), where the real and imaginary components are used as separate real inputs for the activation function. This approach avoids the unbounded nature of the nonlinear complex function but results in a nowhere analytic complex function, as the Cauchy-Riemann equations are not satisfied (Saff and Snider 2003).

Kim and Adali (2002) proposed a set of elementary activation functions with the property of being *almost everywhere* (*a.e.*) bounded and analytic in the complex domain. The complex hyperbolic tangent, $\tanh(z)$, is among them, provided the complex optimization is performed with certain constraints on z. If the magnitude of z is within a circle of radius $\frac{\pi}{2}$, then the singularities do not pose any problem, and the boundedness property is also satisfied. In reality, the dot product of the input and weight vectors may be $\geq \frac{\pi}{2}$. Thus a restriction on the magnitudes of the input and weights is needed.

The NLCPCA model proposed by Rattan and Hsieh (2004, 2005) uses basically the same architecture (Fig. 8.1a) as the NLPCA model of Kramer (1991), except all the input and output variables, and the weight and offset parameters are now complex-valued. The magnitude of input data are scaled by dividing each element in the rth row of the $l \times n$ data matrix $\mathbf{Z}$ (with l the number of variables and n the number of observations) by the maximum magnitude of an element in that row, so each element of $\mathbf{Z}$ has magnitude ≤ 1. The weights at the first hidden layer are randomly initalized with small magnitude, thus limiting the magnitude of the dot product between the input vector and weight vector to be about 0.1, and a weight penalty term is added to the objective function J to restrict the weights to small magnitude during optimization. The weights at subsequent layers are also randomly initialized with small magnitude and penalized during

optimization by the objective function

$$J = \langle \|\mathbf{z} - \mathbf{z}'\|^2 \rangle + P \sum_j |w_j|^2, \tag{8.25}$$

where $\mathbf{z}$ is the model output, $\mathbf{z}'$, the target data, w_j, the individual weights from hidden layers 1, 2 and 3, and P, the weight penalty parameter.

Since the objective function J is a real function with complex weights, the optimization of J is equivalent to finding the vanishing gradient of J with respect to the real and the imaginary parts of the weights (Rattan and Hsieh 2005). All the weights (and offsets) in the model are combined into a single weight vector $\mathbf{w}$. Hence the gradient of the objective function with respect to the complex weights can be split into (Georgiou and Koutsougeras 1992):

$$\frac{\partial J}{\partial \mathbf{w}} = \frac{\partial J}{\partial \mathbf{w}^{\mathrm{R}}} + i \frac{\partial J}{\partial \mathbf{w}^{\mathrm{I}}} \tag{8.26}$$

where $\mathbf{w}^{\mathrm{R}}$ and $\mathbf{w}^{\mathrm{I}}$ are the real and the imaginary components of the weight vector. The two components can be put into a single real parameter vector during nonlinear optimization using an algorithm for real variables.

The tropical Pacific wind anomalies (expressed as $w = u + iv$) have been analyzed by NLCPCA in Rattan and Hsieh (2004), where a comparison between the first mode of CPCA and that of NLCPCA revealed a large difference in the spatial anomaly patterns during strong El Niño episodes but a much smaller difference during strong La Niña episodes, indicating stronger nonlinearity was manifested in the El Niño side than the La Niña side of the oscillation.

The second type of NLCPCA application is for nonlinear Hilbert PCA. In Rattan et al. (2005), evolution of the offshore bottom topography at three sandy barred beaches were studied. All three sites were characterized by sandbars with interannual quasi-periodic offshore propagation. A bar cycle comprises bar birth in the inner nearshore, followed by up to several years of net offshore migration and final disappearance in the outer nearshore zone. CPCA was applied to the complexified topographic anomaly data, and the five leading CPCs were retained as inputs for the NLCPCA NN model. The first NLCPCA mode and the first CPCA mode of the topographic anomalies at Egmond aan Zee (The Netherlands) were compared. The topographic anomalies reconstructed from the nonlinear and linear mode were divided in 8 θ classes, each $\pi/4$ in width, where θ is the phase of the (nonlinear or linear) complex PC. Figure 8.11 shows how the shape of the topographic anomalies change with phase. The CPCA shows sinusoidal-shaped topographic anomalies propagating offshore, while the NLCPCA shows non-sinusoidal anomalies – relatively steep sandbars and shallow, broad troughs. The percentage variance explained by the first NLCPCA mode was 81.4% versus 66.4% by the first CPCA mode. Thus, using the NLCPCA as nonlinear Hilbert PCA successfully captures the non-sinusoidal wave properties which were missed by the linear method.

8.7 Summary and Discussion

The nonlinear generalization of the classical PCA method has been achieved by a number of different approaches (neural networks, principal curves, kernel methods, etc.). We have presented nonlinear PCA (NLPCA) using neural network methods. The tropical Pacific SST example illustrates that with a complicated oscillation like the El Niño-La Niña phenomenon, using a linear method such as PCA results in the nonlinear mode being scattered into several linear modes. In the study of climate variability, the wide use of PCA methods has created the somewhat misleading view that our climate is dominated by a number of spatially fixed oscillatory patterns, which may in fact be due to the limitation of the linear method.

By using a curve instead of a straight line to fit the data, NLPCA is susceptible to overfitting, yielding excessively wiggly curves. The introduction of a weight penalty parameter P in the objective function allowed the wiggles to be smoothed, but the lack of an objective selection criterion for P (hence the amount of smoothing) has been a weakness in NLPCA, until recent advances have allowed the objective selection of P and m (which controls the number of hidden neurons, hence model complexity). While the information criterion introduced by Hsieh (2007) for model selection worked well in climate datasets where there is one dominant signal (e.g. ENSO in the tropical Pacific SST; QBO in the stratospheric wind), it remains inadequate for dealing with datasets which contain two or more distinct signals of roughly comparable strength – e.g. in the extratropical N. Hemisphere climate, where there has been considerable controversy on the use of

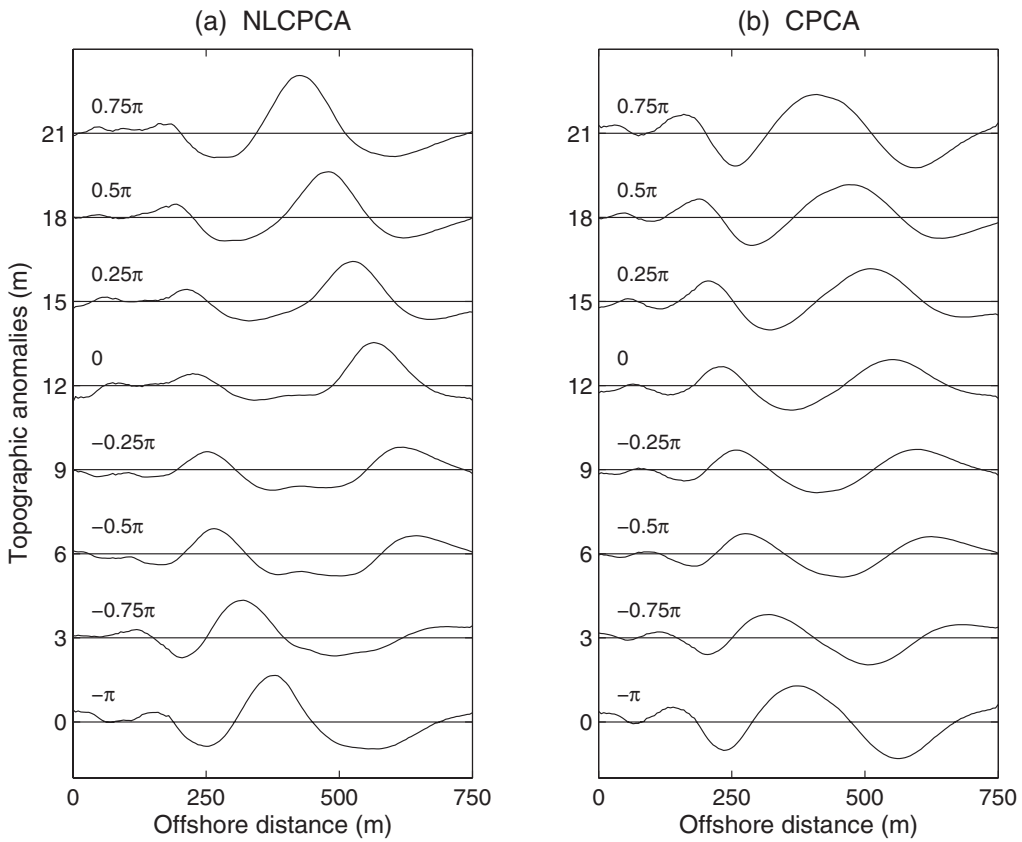


Fig. 8.11 Sequence of topographic anomalies as a function of the offshore distance at Egmond in $\pi/4$-wide θ classes centered around $\theta = -\pi$ to $\theta = 0.75\pi$ based on (a) NLCPCA mode 1 and (b) CPCA mode 1. The results for each phase class have been vertically shifted for better visualization. The phase generally decreases with time, so the anomalies gradually propagate offshore (Modified from Rattan et al. 2005)

NLPCA (Christiansen 2005, 2007; Monahan and Fyfe 2007), there are two signals of comparable magnitude, the Arctic Oscillation and the Pacific-North American teleconnection. The reason is that if there are two comparable signals, the total signal forms a 2-D surface whereas the NLPCA model will be trying to fit a 1-D curve to this surface, resulting in a hybrid mode with attributes from both signals. While it is possible to have two neurons in the bottleneck layer in the NLPCA network, so that a 2-D solution is extracted, there is no simple way to separate the two signals. Clearly more research is needed in developing model selection criteria in NLPCA for such complicated noisy datasets.

NLPCA has also been generalized to closed curve solutions, and to complex variables. Self-organizing maps (SOM) provide a discrete version of NLPCA. Due to space limitation, further generalization to nonlinear singular spectrum analysis and nonlinear canonical correlation analysis have not been presented here (see the review by Hsieh 2004).

References

Baldwin, M., Gray, L., Dunkerton, T., Hamilton, K., Haynes, P., Randel, W., Holton, J., Alexander, M., Hirota, I., Horinouchi, T., Jones, D., Kinnersley, J., Marquardt, C., Sato, K., & Takahashi, M. (2001). The quasi-biennial oscillation. *Reviews of Geophysics*, *39*, 179–229.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. (482 pp.) Oxford: Oxford University Press.

Cavazos, T. (1999). Large-scale circulation anomalies conducive to extreme precipitation events and derivation of daily rainfall in northeastern Mexico and southeastern Texas. *Journal of Climate*, *12*, 1506–1523.

Cavazos, T. (2000). Using self-organizing maps to investigate extreme climate events: An application to wintertime precipitation in the Balkans. *Journal of Climate*, *13*, 1718–1732.

Cavazos, T., Comrie, A. C., & Liverman, D. M. (2002). Intraseasonal variability associated with wet monsoons in southeast Arizona. *Journal of Climate*, *15*, 2477–2490.

Cherkassky, V., & Mulier, F. (1998). *Learning from data* (441 pp.). New York: Wiley.

Christiansen, B. (2005). The shortcomings of nonlinear principal component analysis in identifying circulation regimes. *Journal of Climate*, *18*, 4814–4823.

Christiansen, B. (2007). Reply to Monahan and Fyfe's comment on "The shortcomings of nonlinear principal component analysis in identifying circulation regimes". *Journal of Climate*, *20*, 378–379. DOI: 10.1175/JCLI4006.1.

Clarke, T. (1990). Generalization of neural network to the complex plane. *Proceedings of International Joint Conference on Neural Networks*, *2*, 435–440.

Del Frate, F., & Schiavon, G. (1999). Nonlinear principal component analysis for the radiometric inversion of atmospheric profiles by using neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, *37*, 2335–2342.

Diaz, H. F., & Markgraf, V. (Eds.) (2000) *El Nino and the southern oscillation: Multiscale variability and global and regional impacts* (496 pp.). Cambridge: Cambridge University Press.

Georgiou, G., & Koutsougeras, C. (1992). Complex domain backpropagation. *IEEE Transactions on Circults and Systems II*, *39*, 330–334.

Hamilton, K. (1998). Dynamics of the tropical middle atmosphere: A tutorial review. *Atmosphere-Ocean*, *36*, 319–354.

Hamilton, K., & Hsieh, W. W. (2002). Representation of the QBO in the tropical stratospheric wind by nonlinear principal component analysis. *Journal of Geophysical Research*, *107*. DOI: 10.1029/2001JD001250.

Hardman-Mountford, N. J., Richardson, A. J., Boyer, D. C., Kreiner, A., & Boyer, H. J. (2003). Relating sardine recruitment in the Northern Benguela to satellite-derived sea surface height using a neural network pattern recognition approach. *Progress in Oceanograply*, *59*, 241–255.

Hastie, T., & Stuetzle, W. (1989). Principal curves. *Journal of the American Statistical Association*, *84*, 502–516.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *Elements of statistical learning: Data mining, inference and prediction* (552 pp.). New York: Springer.

Hirose, A. (1992). Continuous complex-valued backpropagation learning. *Electronic Letters*, *28*, 1854–1855.

Hoerling, M. P., Kumar, A., & Zhong, M. (1997). El Nino, La Nina and the nonlinearity of their teleconnections. *Journal of Climate*, *10*, 1769–1786.

Holton, J. R., & Tan, H.-C. (1980). The influence of the equatorial quasi-biennial oscillation on the global circulation at 50 mb. *Journal of the Atmospheric Sciences*, *37*, 2200–2208.

Hsieh, W. W. (2001). Nonlinear principal component analysis by neural networks. *Tellus*, *53*A, 599–615.

Hsieh, W. W. (2004). Nonlinear multivariate and time series analysis by neural network methods. *Reviews of Geophysics*, *42*, RG1003. DOI: 10.1029/2002RG000112.

Hsieh, W. W. (2007). Nonlinear principal component analysis of noisy data. *Neural Networks*, *20*, 434–443. DOI 10.1016/j.neunet.2007.04.018.

Hsieh, W. W., & Wu, A. (2002). Nonlinear multichannel singular spectrum analysis of the tropical Pacific climate variability using a neural network approach. *Journal of Geophysical Research*, *107*. DOI: 10.1029/2001JC000957.

Jolliffe, I. T. (2002). *Principal component analysis* (502 pp.) Berlin: Springer.

Kaiser, H. F. (1958). The varimax criterion for analytic rotation in factor analysis. *Psychometrika*, *23*, 187–200.

Kim, T., & Adali, T. (2002). Fully complex multi-layer perceptron network for nonlinear signal processing. *Journal of VLSI Signal Processing*, *32*, 29–43.

Kirby, M. J., & Miranda, R. (1996). Circular nodes in neural networks. *Neural Computation*, *8*, 390–402.

Kohonen, T. (1982). Self-organzing formation of topologically correct feature maps. *Biological Cybernetics*, *43*, 59–69.

Kohonen, T. (2001). *Self-Organizing maps* (3rd ed., 501 pp.) Berlin: Springer.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, *37*, 233–243.

Liu, Y., Wieisberg, R. H., & Mooers, C. N. K. (2006). Performance evaluation of the self-organizing map for feature extraction. *Journal of Geophysical Research*, *111*. DOI: 10.1029/2005JC003117.

Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, *20*, 130–141.

Monahan, A. H. (2000). Nonlinear principal component analysis by neural networks: Theory and application to the Lorenz system. *Journal of Climate*, *13*, 821–835.

Monahan, A. H. (2001). Nonlinear principal component analysis: Tropical Indo-Pacific sea surface temperature and sea level pressure. *Journal of Climate*, *14*, 219–233.

Monahan, A. H., & Fyfe, J. C. (2007). Comment on "The shortcomings of nonlinear principal component analysis in identifying circulation regimes". *Journal of Climate*, *20*, 375–377. DOI: 10.1175/JCLI4002.1.

Monahan, A. H., Fyfe, J. C., & Flato, G. M. (2000). A regime view of northern hemisphere atmospheric variability and change under global warming. *Geophysics Research Letters*, *27*, 1139–1142.

Monahan, A. H., Pandolfo, L., & Fyfe, J. C. (2001). The preferred structure of variability of the northern hemisphere atmospheric circulation. *Geophysical Research Letters*, *28*, 1019–1022.

Newbigging, S. C., Mysak, L. A., & Hsieh, W. W. (2003). Improvements to the non-linear principal component analysis method, with applications to ENSO and QBO. *Atmosphere-Ocean*, *41*, 290–298.

Nitta, T. (1997). An extension of the back-propagation algortihm to complex numbers. *Neural Networks*, *10*, 1391–1415.

Oja, E. (1982). A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, *15*, 267–273.

Philander, S. G. (1990). *El Niño, La Niña, and the southern oscillation* (293 pp.). San Diego, CA: Academic.

Preisendorfer, R. W. (1988). *Principal component analysis in meteorology and oceanography* (425 pp.). Amsterdam: Elsevier.

Rattan, S. S. P., & Hsieh, W. W. (2004), Nonlinear complex principal component analyis of the tropical Pacific interannual wind variability. *Geophysical Research Letters*, *31* (21), L21201. DOI: 10.1029/2004GL020446.

Rattan, S. S. P., & Hsieh, W. W. (2005). Complex-valued neural networks for nonlinear complex principal component analysis. *Neural Networks*, *18*, 61–69. DOI: 10.1016/j.neunet.2004.08.002.

Rattan, S. S. P., Ruessink, B. G., & Hsieh, W. W. (2005). Nonlinear complex principal component analysis of nearshore bathymetry. *Nonlinear Processes in Geophysics*, *12*, 661–670.

Richardson, A. J., Risien, C., & Shillington, F. A. (2003). Using self-organizing maps to identify patterns in satellite imagery. *Progress in Oceanography*, *59*, 223–239.

Richman, M. B. (1986). Rotation of principal components. *Journal of Climatology*, *6*, 293–335.

Rojas, R. (1996). *Neural networks – A systematic introduction* (502 pp.). Berlin: Springer.

Ruessink, B. G., van Enckevort, I. M. J., & Kuriyama, Y. (2004). Non-linear principal component analysis of nearshore bathymetry. *Marine Geology*, *203*, 185–197.

Saff, E. B., & Snider, A. D. (2003). *Fundamentals of complex analysis with applications to engineering and science* (528 pp.). Englewood Cliffs, NJ: Prentice-Hall.

Sanger, T. D. (1989). Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, *2*, 459–473.

Schölkopf, B., Smola, A., & Muller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, *10*, 1299–1319.

Tang, Y., & Hsieh, W. W. (2003). Nonlinear modes of decadal and interannual variability of the subsurface thermal structure in the Pacific Ocean. *Journal of the Geophysical Research*, *108*. DOI: 10.1029/2001JC001236.

Villmann, T., Merenyi, E., & Hammer, B. (2003). Neural maps in remote sensing image analysis. *Neural Networks*, *16*, 389–403.

von Storch, H., & Zwiers, F. W. (1999). *Statistical analysis in climate research* (484 pp.). Cambridge: Cambridge University Press.

Webb, A. R. (1999). A loss function approach to model selection in nonlinear principal components. *Neural Networks*, *12*, 339–345.

Yacoub, M., Badran, F., & Thiria, S. (2001). A topological hierarchical clustering: Application to ocean color classification. *Artificial Neural Networks-ICANN 2001, Proceedings. Lecture Notes in Computer Science*, 492–499.

Neural Network Applications to Solve Forward and Inverse Problems in Atmospheric and Oceanic Satellite Remote Sensing

9

Vladimir M. Krasnopolsky

List of Acronyms

BT – Brightness Temperature
DAS – Data Assimilation System
FM – Forward Model
MLP – Multi-Layer Perceptron
NWP – Numerical Weather Prediction
PB – Physically Based
RMSE – Root Mean Square Error
RS – Remote Sensing
SSM/I – Special Sensor Microwave Imager
SST – Sea Surface Temperature
TF – Transfer Function

9.1 Introduction

Here we discuss two very important practical applications of the neural network (NN) technique: solution of forward and inverse problems in atmospheric and oceanic satellite remote sensing (RS). A particular example of this type of NN applications – solving the SAR wind speed retrieval problem – is also presented in Chapter 10 by G. Yung. These applications and those that we discuss in Chapter 11, from the mathematical point of view, belong to the broad class of applications called approximation of mappings. A particular type of the NN, a Multi-Layer Perceptron (MLP) NN (Rumelhart et al. 1986) is usually employed to approximate mappings. We will start by introducing a remote sensing, mapping, and NN background.

9.1.1 Remote Sensing Background

Estimating high quality geophysical parameters (information about the physical, chemical, and biological properties of the oceans, atmosphere, and land surface) from remote measurements (satellite, aircraft, etc.) is a very important problem in fields such as meteorology, oceanography, climatology and environmental modeling and prediction. Direct measurements of many parameters of interest, like vegetation moisture, phytoplankton concentrations in the ocean, and aerosol concentrations in the atmosphere are, in general, not available for the entire globe at the required spatial and temporal resolution. Even when *in situ* measurements are available, they are usually sparse (especially over the oceans) and located mainly at ground level or at the ocean surface. Often such measurements can be estimated indirectly from the influence of these geophysical parameters on the electromagnetic radiation measured by a remote sensor. Remote measurements allow us to obtain spatially dense measurements all over the globe at and above the level of the ground and ocean surface.

Vladimir M. Krasnopolsky (✉)
Science Application International Company at Environmental Modeling Center, National Centers for Environmental Prediction, National Oceanic and Atmospheric Administration, Camp Springs, Maryland, USA

Earth System Science Interdisciplinary Center, University of Maryland, EMC/NCEP/NOAA, 5200 Auth Rd., Camp Springs, MD 20746, USA
Phone: 301-763-8000 ext. 7262; fax 301-763-8545;
email: vladimir.krasnopolsky@noaa.gov

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

Satellite RS data are used in a wide variety of applications and by a wide variety of users. Satellite sensors generate measurements like radiances, backscatter coefficients, brightness temperatures, etc. The applications usually utilize geophysical parameters such as pressure, temperature, wind speed and direction, water vapor concentration, etc. derived from satellite data. Satellite forward models, which simulate satellite measurements from given geophysical parameters, and retrieval algorithms, which transform satellite measurements into geophysical parameters, play the role of mediators between satellite sensors and applications. There exists an entire spectrum of different approaches in extracting geophysical information from the satellite measurements. At one end of this spectrum 'satellite only' approaches are located; we will call them standard or traditional retrievals. They use measurements performed by one particular sensor only, sometimes from different channels (frequencies, polarizations, etc.) of the same sensor to estimate geophysical parameters. Variational retrieval techniques or direct assimilation techniques are located at the other end of the spectrum. They use an entire data assimilation system (DAS), including a numerical weather prediction (NWP) model and analysis (Prigent et al. 1997), which in turn includes all kind of meteorological measurements (buoys, radiosondes, ships, aircrafts, etc.) as well as data from numerous satellite sensors. Data assimilation is a method in which observations of the current (and possibly, past) state of a system (atmosphere and/or ocean) are combined with the results from a numerical model to produce an *analysis*, which is considered as 'the best' estimate of the current state of the system. The analysis can be used for many purposes including initialization of the next step of the numerical model integration. Many approaches have been developed which belong to the intermediate part of this spectrum. These approaches use measurements from several satellite sensors, combine satellite measurements with other kinds of measurements, and/or use background fields or profiles from NWP models for regularization of the inverse problem (retrievals) or for ambiguity removal, i.e., these approaches use some type of data fusion to regularize (see Sections 9.1.2 and 9.4 below) the solution of the inverse problem.

It is noteworthy that over the last few years, direct assimilation of some geophysical parameters into modern DASs has been successfully developed and implemented. It improved the quality of assimilated products and numerical forecasts that use some of these products as initial conditions. Direct assimilation replaces or eliminates the need for using retrievals of these geophysical parameters in DASs. However, there are still many other geophysical parameters (e.g., precipitations, atmospheric ice) that have not yet been included, or it is not clear from both theoretical and/or practical considerations how they could be included into DASs through direct assimilation. There are also other users of the retrieved geophysical parameters. Therefore, there is still an urgent need to use the standard retrievals for these geophysical parameters and to develop the corresponding retrieval algorithms to which the NN technique could be efficiently applied. Direct assimilation is discussed below at the end of this subsection in the description of variational retrieval techniques.

The remote measurements themselves are usually very accurate. The quality of geophysical parameters derived from these measurements varies significantly depending on the strength and uniqueness of the signal from the geophysical parameter and the mathematical methods applied to extract the parameter, i.e., to solve RS forward and/or inverse problems (see Section 9.2). The NN technique is a useful mathematical tool for solving the forward and inverse problems in RS accurately. The number of NN RS applications has been increasing steadily over the last decade.

A broad class of NN applications has been developed for solving the forward and inverse problems in RS in order to infer geophysical parameters from satellite data, i.e., to produce so-called satellite retrievals. A brief review of RS NN applications was presented by Atkinson and Tatnall (1997). Examples of such applications follow. The NN technique was applied for the inversion of a multiple scattering model to estimate snow parameters from passive microwave measurements (Tsang et al. 1992). Smith (1993) used NNs for the inversion of a simple two-stream radiative transfer model to derive the leaf area index from Moderate Resolution Imaging Spectrometer data. In other studies, NNs were applied to simulate scatterometer measurements and to retrieve wind speed and direction from these measurements (Thiria et al. 1993; Cornford et al. 2001); to develop an inversion algorithm for radar scattering from vegetation canopies (Pierce et al. 1994); to estimate atmospheric humidity profiles (Cabrera-Mercader and Staelin 1995), atmospheric

temperature, moisture, and ozone profiles (Aires et al. 2002) and atmospheric ozone profiles (Mueller et al. 2003). Stogryn et al. (1994), and Krasnopolsky et al. (1995) applied NNs to invert Special Sensor Microwave Imager (SSM/I) data and retrieve surface wind speed. Davis et al. (1995) applied NNs to invert a forward model to estimate soil moisture, surface air temperature, and vegetation moisture from Scanning Multichannel Microwave Radiometer data. Using a NN technique, a fast SSM/I forward model (Krasnopolsky 1997) and SSM/I multi-parameter retrieval algorithm (Krasnopolsky et al. 1999, 2000; Meng et al. 2007) have been derived from empirical data (buoy SSM/I collocations). Abdelgadir et al. (1998) applied NNs to the forward and inverse modeling of canopy directional reflectance. Schiller and Doerffer (1999) used a NN technique for inverting a radiative transfer forward model to estimate the concentration of phytoplankton pigment from Medium Resolution Imaging Spectrometer data.

9.1.2 Mapping and Neural Networks Background

A mapping, M, between two vectors X (input vector) and Y (output vector) can be symbolically written as,

$$Y = M(X); \quad X \in \Re^n, \; Y \in \Re^m \tag{9.1}$$

where $X \in \Re^n$ means that the vector X has n components and all of them are real numbers. A large number of important practical geophysical applications may be considered mathematically as a mapping like (9.1). Keeping in mind that a NN technique will be used to approximate this mapping, we will call it a **target mapping**, using a common term from nonlinear approximation theory (DeVore 1998). The target mapping may be given to us explicitly or implicitly. It can be given explicitly as a set of equations based on first principles (e.g., radiative transfer or heat transfer equations) and/or empirical dependencies, or as a computer code. Observational records composed of Xs and Ys represent an implicit target mapping. In this case, it is assumed that the unknown target mapping generates the data.

The mapping (9.1) is a complicated mathematical object with many important characteristics like mapping dimensionalities, domain, range, complexities, etc. Some of them are illustrated by Fig. 9.1.

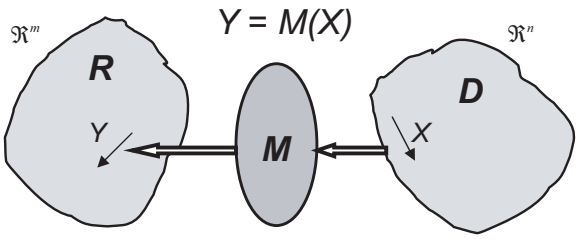


Fig. 9.1 The mapping (1), M; its input vector, X; output vector, Y; domain, D and range, R

Among the applications considered in this chapter, we will find inverse problems that can be considered as continuous unique mappings (9.1); however, for these mappings small perturbations in X may cause large changes in Y and the problem is then called ill-posed (Vapnik 1995). Large changes in Y may occur because an ill-posed problem may have more than one solution, or the solution may depend discontinuously upon the initial data (Hadamard 1902). It is also known as improperly posed problem. Ill-posed problems usually arise when one attempts *to estimate an unknown cause from observed effects* (most of the geophysical inverse problems belong to this class, e.g., the satellite retrieval problem considered in Section 9.2.1) or *to restore a whole object from its low dimensional projection* (e.g., estimating the NN Jacobian considered in Aires et al. 2004 and Krasnopolsky 2006). If X contains even a low level of noise, the uncertainties in Y may be very large. To solve ill-posed problems additional *a priori* information about the solution (regularization) should be introduced into the solution approach (Vapnik and Kotz 2006).

The simplest MLP NN is a generic analytical nonlinear approximation or model for mapping, like the target mapping (9.1). The MLP NN uses for the approximation a family of functions like:

$$y_q = NN(X, a, b) = a_{q0} + \sum_{j=1}^{k} a_{qj} \cdot z_j; \quad q = 1, 2, \ldots, m \tag{9.2}$$

$$z_j = \phi(b_{j0} + \sum_{i=1}^{n} b_{ji} \cdot x_i) \tag{9.3}$$

where x_i and y_q are components of the input and output vectors respectively, a and b are fitting parameters, and ϕ is a so called activation function (a nonlinear

function, often a hyperbolic tangent), n and m are the numbers of inputs and outputs respectively, and k is the number of the nonlinear basis function z_j (9.3) in the expansion (9.2). The expansion (9.2) is a linear expansion (a linear combination of the basis function z_j (9.3)) and the coefficients a_{qj} ($q = 1, \ldots, m$ and $j = 1, \ldots, k$) are linear coefficients in this expansion. It is essential that the basis functions z_j (9.3) are nonlinear with respect to inputs x_i ($i = 1, \ldots, n$) and to the fitting parameters or coefficients b_{ji} ($j = 1, \ldots, k$). As a result of the nonlinear dependence of the basis functions on multiple fitting parameters b_{ji}, the basis $\{z_j\}_{j=1,\ldots,k}$ turns into a very flexible set of non-orthogonal basis functions that have a great potential to adjust to the functional complexity of the mapping (9.1) to be approximated. It has been shown by many authors in different contexts that the family of functions (9.2, 9.3) can approximate any continuous or almost continuous (with a finite number of finite discontinuities, like a step function) mapping (9.1) (Cybenko 1989; Funahashi 1989; Hornik 1991; Chen and Chen 1995a, b). The accuracy of the NN approximation or the ability of the NN to resolve details of the target mapping (9.1) is proportional to the number of basis functions (hidden neurons) k (Attali and Pagès 1997).

In this chapter and in Chapter 11, we use the terms an **emulating NN** or a **NN emulation** for NN (9.2, 9.3) that provides a functional emulation of the target mapping (9.1) that implies a small approximation error for the training set and smooth and accurate interpolation between training set data points inside the mapping domain D. The term "emulation" is introduced to distinguish between these NNs and approximating NNs or NN approximations that guarantee small approximation error for the training set only.

When an emulating NN is developed, in addition to the criterion of small approximation error at least three other criteria are used: (i) the NN complexity (proportional to the number k of hidden neurons when other topological parameters are fixed) is controlled and restricted to a minimal level sufficient for good approximation and interpolation; (ii) independent validation and test data sets are used in the process of training (validation set) to control overfitting and after the training (test set) to evaluate interpolation accuracy; (iii) redundant training set (additional redundant data points are added in-between training data points sufficient for a good approximation) is used for improving the NN interpolation abilities (Krasnopolsky 2007).

9.2 Deriving Geophysical Parameters from Satellite Measurements: Standard Retrievals and Variational Retrievals Obtained Through Direct Assimilation

9.2.1 Standard or Conventional Retrievals

Conventional methods for using satellite data (standard retrievals) involve solving an inverse or retrieval problem and deriving a transfer function (TF) f, which relates a geophysical parameter of interest G (e.g., surface wind speed over the ocean, atmospheric moisture concentration, sea surface temperature (SST), etc.) to a satellite measurement S (e.g., brightness temperatures, radiances, reflection coefficients, etc.)

$$G = f(S) \tag{9.4}$$

where both G and S may be vectors. The TF f, (also called a retrieval algorithm) usually cannot be derived directly from first principles because the relationship (9.4) does not correspond to a cause and effect principle and multiple values of G can sometimes correspond to a single S. Forward models (FM),

$$S = F(G) \tag{9.5}$$

where F is a forward model, which relate a vector G to a vector S, can usually be derived from first principles and physical considerations (e.g., a radiative transfer theory) in accordance with cause and effect principles because geophysical parameters affect the satellite measurements (but not vice versa). Thus, the forward problem (9.5) is a well-posed problem in contrast to the inverse problem (9.4) which is often an ill-posed one (Parker 1994); although, from a mathematical point of view, both FM (9.5) and TF (9.4) are continuous (or almost continuous) mappings between the two vectors S and G. Even in the cases where the mapping (9.4) is not unique, this multi-valued mapping may be considered as a collection of single-valued continuous mappings. In order to derive the TF (9.4), the FM (9.5) has to be inverted (an inverse problem has to be

solved). The inversion technique usually searches for a vector G^0 which minimizes the functional (Stoffelen and Anderson 1997)

$$\|\Delta S\| = \left\| S^0 - F(G) \right\| \tag{9.6}$$

where S^0 is an actual vector of satellite measurements. Since the FM F is usually a complicated nonlinear function, this approach leads to a full-scale nonlinear optimization with all its numerical problems, like slow convergence, multiple solutions, etc. This approach does not determine the TF explicitly; it assumes this function implicitly, and for each new measurement S^0 the entire process has to be repeated. A simplified linearization method to minimize the functional (9.6) can be applied if there is a good approximation for the solution of the inverse problem, that is, an approximate vector of the geophysical parameters G^0. Then the difference vector ΔS is small and there is a vector G in close proximity to G^0 ($|\Delta G| = |G - G^0|$ is small) where $\Delta S(G) = 0$. Expanding $F(G)$ in a Taylor series and keeping only those terms which are linear with respect to ΔG, we can obtain a system of linear equations to calculate the components of the vector ΔG (e.g., Wentz 1997),

$$\sum_{i=1}^{n} \frac{\partial F(G)}{\partial G_i}|_{G=G^0} \Delta G_i = S^0 - F(G^0) \tag{9.7}$$

where n is the dimension of vector G. After ΔG is calculated, the next iteration of (9.7) with $G^0 = G^0 + \Delta G$ is performed. The process is expected to converge quickly to the vector of retrievals G. Again, in this case the TF, f, is not determined explicitly but is only determined implicitly for the vector S^0 by the solution of (9.7). This type of retrieval can be called a "local" or "localized" linear inversion. These techniques (9.6, 9.7) are usually called physically based retrievals. It is important to emphasize that the physically based algorithms (9.6, 9.7) are by definition multi-parameter algorithms since they retrieve several geophysical parameters simultaneously (a complete vector G).

Empirical algorithms are based on an approach which, from the beginning, assumes the existence of an explicit analytical representation for a TF, f. A mathematical (statistical) model, f_{mod}, is usually chosen (usually some kind of a regression), which contains a vector of fitting (or regression) parameters $a = \{a_1,\ a_2, \ldots\}$,

$$G_k = f_{\text{mod}}(S, a) \tag{9.8}$$

where these parameters are determined from an empirical (or simulated) matchup data set $\{G_k,\ S\}$ collocated in space and time and use, for example, statistical techniques such as the method of least-squares. This type of retrieval can be called a "global" inversion as it is not restricted to a given vector of satellite measurements. The subscript k in G_k stresses the fact that the majority of empirical retrieval algorithms are single-parameter algorithms. For example, for Special Sensor Microwave Imager (SSM/I) there exist algorithms which retrieve only wind speed (Goodberlet et al. 1989), water vapor (Alishouse et al. 1990; Petty 1993), or cloud liquid water (Weng and Grody 1994). Krasnopolsky et al. (1999, 2000) showed that single-parameter algorithms have additional (compared to multi-parameter retrievals) systematic (bias) and random (unaccounted variance) errors in a single retrieved parameter G_k.

The obvious way to improve single-parameter retrievals (9.8) is to include other parameters in the retrieval process using an empirical multi-parameter approach, which as in the physically based multi-parameter approach (9.6, 9.7), inverts the data in the complete space of the geophysical parameters (Krasnopolsky et al. 1999, 2000). Thus, the complete vector of the related geophysical parameters is retrieved simultaneously from a given vector of satellite measurements S,

$$G = f_{\text{mod}}(S, a) \tag{9.9}$$

where $G = \{G_i\}$ is now a vector containing the primary, physically-related geophysical parameters, which contribute to the observed satellite measurements S. These retrievals do not contain the additional systematic and random errors just described. Because equations (9.4), (9.5), (9.8), and (9.9) represent continuous mappings, the NN technique is well suited for emulating the FM, TF, and f_{mod}.

The standard retrievals derived using TF (9.4) have the same spatial resolution as the sensor measurements and produce instantaneous values of geophysical parameters over the areas where the measurements are available. Geophysical parameters derived using standard retrievals can be used for many applications, such as the NWP DASs. In this case, a contribution to the variational analysis cost function χ_G from a particular

retrieval, G^0, is:

$$\chi_G = \frac{1}{2}(G - G^0)^T (O + E)^{-1} (G - G^0) \qquad (9.10)$$

where $G^0 = f(S^0)$ is a vector of the retrieved geophysical parameter, S^0 is a vector of the sensor measurements, G is a vector of the geophysical parameters being analyzed, O is the expected error covariance of the observations, and E is the expected error covariance of the retrieval algorithm.

9.2.2 Variational Retrievals Through the Direct Assimilation of Satellite Measurements

Because standard retrievals are based on the solution of an inverse problem which is usually mathematically ill-posed (Parker 1994), this approach has some rather subtle properties and error characteristics (Eyre and Lorenc 1989) which cause additional errors and problems in retrievals (e.g., an amplification of errors, ambiguities, etc.). As a result, high-quality sensor measurements might be converted into lower-quality geophysical parameters. This type of error can be avoided or reduced by using a variational retrieval technique (or an inversion) through direct assimilation of satellite measurements (Lorenc 1986; Parrish and Derber 1992; Phalippou 1996; Prigent et al. 1997; Derber and Wu 1998; McNally et al. 2000).

Variational retrievals or direct assimilation of satellite data offer an alternative to deriving geophysical parameters from the satellite measurements. They use the entire data assimilation system for the inversion (as a retrieval algorithm). In this case, a contribution to the analysis cost function χ_S from a particular sensor measurement, S^0, is:

$$\chi_S = \frac{1}{2}(S - S^0)^T (O + E)^{-1} (S - S^0) \qquad (9.11)$$

where $S = F(G)$, and F is a FM which relates an analysis state vector G (or a vector of geophysical parameters in the analysis) to a vector of simulated sensor measurements S, O is the expected error covariance of the observations, and E is the expected error covariance of the forward model. The forward problem (9.5) is a well-posed problem in contrast to the inverse problem (9.4). However, a background term has to be added to (9.11) to prevent the data assimilation problem from being ill-posed (Parrish and Derber 1992).

The retrieval in this case results in an entire field (global in the case of the global data assimilation system) for the geophysical parameter G (non-local retrievals) which has the same resolution as the numerical forecast model used in the data assimilation system. This resolution may be lower or higher than the resolution of standard retrievals. The variational retrievals are also not instantaneous but usually averaged in time over the analysis cycle; however, the field is continuous and coherent (e.g., it should not have problems such as directional ambiguity in the scatterometer winds). The variational retrievals are the result of fusing many different types of data (including satellite data, ground observations, and numerical model first guesses) inside the data assimilation system. Sparse standard retrievals can be converted into continuous fields using the regular data assimilation procedure (9.10) that fuses sparse observations with the numerical model short term prediction producing the integrated product on the model grid.

It is important to emphasize a very significant difference between the use of the explicit TF for standard retrievals and the use of FM in variational retrievals. In standard retrievals, the explicit TF (9.4) is usually simple (e.g., a regression) and is applied once per sensor observation to produce a geophysical parameter. In variational retrievals the FM, which is usually much more complicated than a simple explicit TF, and its partial derivatives (the number of derivatives is equal to $m \times n$, where m and n are the dimensions of the vectors G and S, respectively) have to be estimated for each of the k iterations performed during the cost function (9.11) minimization. Thus the requirements for simplicity of the FM used in the variational retrievals are restrictive, and variational retrievals often require some special, simplified and fast versions of FMs.

9.3 NNs for Emulating Forward Models

FMs are usually complex due to the complexity of the physical processes which they describe and the complexity of the first principle formalism on which they are based (e.g., a radiative transfer theory). Dependencies of satellite measurements on geophysical parameters, which FMs describe, are complicated

and nonlinear. These dependencies may exhibit different types of nonlinear behavior. FMs are usually exploited in physically based retrieval algorithms where they are numerically inverted to retrieve geophysical parameters and in data assimilation systems where they are used for the direct assimilation of satellite measurements (variational retrievals). Both numerical inversions and direct assimilation are iterative processes where FMs and their Jacobians are calculated many times for each satellite measurement. Thus, the retrieval process becomes very time consuming, sometimes prohibitively expensive for operational (real time) applications.

For such applications, it is essential to have fast and accurate versions of FMs. Because the functional complexity of FM mappings (complexity of input/output relationships) is usually not as high as their physical complexity, NNs can provide fast and accurate emulations of FMs. Moreover, a NN can also provide an entire Jacobian matrix with only a small additional computational effort. This is one of NN applications where the NN Jacobian is calculated and used. Because a statistical inference of Jacobian is an ill-posed problem, it should be carefully tested and controlled.

To develop a NN emulation for the FM, a training set which consists of matched pairs of vectors of geophysical parameters and satellite measurements, $\{G, S\}_{i=1,\ldots,N}$, has to be created. If a physically based FM exists, it can be used to simulate the data. Otherwise, empirical data can be used. The resulting data set constitutes the training set and will be employed to develop the NN emulation.

9.4 NNs for Solving Inverse Problems: NNs Emulating Retrieval Algorithms

NNs can be used in several different ways for retrieval algorithms. In physically based retrieval algorithms a fast NN, emulating the complex and slow physically based FM and its Jacobian, can be used to speed up the local inversion process (9.7). NNs can be used in many cases for a global inversion to explicitly invert a FM. In such cases, after an inversion the NN provides an explicit retrieval algorithm (or TF), which is a solution of the inverse problem and can be used for retrievals. To train a NN which emulates an explicit retrieval algorithm, a training set $\{G, S\}_{i=1,\ldots,N}$, is required. As in the case of FMs, simulated or empirical data can be used to create the training set.

In addition to the complications related to FMs (complexity, nonlinearity, etc.), retrieval algorithms exhibit some problems because they are solutions of the inverse problem, which is an ill-posed problem. This is why mathematical tools which are used to develop retrieval algorithms have to be accurate and robust in order to deal with these additional problems. NNs are fast, accurate and robust tools for emulating nonlinear (continuous) mappings and can be effectively used for modeling multi-parameter retrieval algorithms. One of serious problems related to retrieval algorithms is the problem of regularizing the solution of the inverse problem. Without regularization, from very accurate satellite measurements only poor quality or ambiguous geophysical parameters can usually be retrieved. To regularize an ill-posed inverse problem, additional (regularization) information should be introduced (Vapnik and Kotz 2006). The NN technique is flexible enough to accommodate regularization information as additional inputs and/or outputs and as additional regularization terms in the error or loss function. For example, in their pioneering work on using NNs for the simultaneous retrieval of temperature, water vapor, and ozone atmospheric profiles (Aires et al. 2002; Mueller et al. 2003) from satellite measurements, the authors made good use of this NN flexibility by introducing the first guess from the atmospheric model or DAS as additional regularizing inputs in their NN based retrieval algorithms.

9.5 Controlling the NN Generalization

NNs are well suited to modeling complicated nonlinear relationships between multiple variables, as is the case in multispectral remote sensing. Well-constructed NNs (NN emulations) have good interpolation properties; however, they may produce unpredictable outputs when forced to extrapolate. The NN training data (simulated by a FM or constructed from empirical data sets) cover a certain manifold D_T (a sub-domain $D_T \in D$) in the full domain D. Real data to be fed into the NN, f_{NN}, which emulates a TF, may not always lie in D_T. There are many reasons for such deviations of real data from the low dimensional manifold D_T of

training data, e.g., simplifications built into a model design, neglecting the natural variability of parameters occurring in the model and measurement errors in the satellite signal not taken into account during the generation of the training data set. When empirical data are used, extreme events (highest and lowest values of geophysical parameters) are usually not sufficiently represented in the training set because they have a low frequency of occurrence in nature. This means that during the retrieval stage, real data in some cases may force the NN emulation, f_{NN}, to extrapolate. The error resulting from such a forced extrapolation will increase with the distance of the input point from D_T and will also depend on the orientation of the input point relative to D_T.

In order to recognize NN inputs not foreseen in the NN training phase and, thus, out of the scope of the inversion algorithm, a validity check (Schiller and Krasnopolsky 2001) can be used. This check may serve as the basis for a quality control (QC) procedure. Some kind of QC procedure is usually applied to the satellite retrievals in DAS. Let the model $S = F(G)$ have an inverse $G = f(S)$, then, by definition $S = F(f(S))$. Further, let f_{NN} be the NN emulating the inverse model in the domain D_T. The result of $G_0 = f_{NN}(S_0)$ for $S_0 \notin D_T$ may be arbitrary and, in general, $F(f_{NN}(S_0))$ will not be equal to S_0. The validity of $S = F(f_{NN}(S))$ is a necessary condition for $S \in D$. Now, if in the application stage of the NN, f_{NN}, S is not in the domain S_T, the NN is forced to extrapolate. In such a situation the validity condition may not be fulfilled, and the resulting G is in general meaningless. For operational applications, it is necessary to report such events to make a proper decision about correcting this retrieval or removing it from the data stream. In order to perform the validity test, the FM must be applied after each inversion. This requires a fast but accurate FM. Such a FM can be achieved by developing a NN that accurately emulates the original FM, $S = F_{NN}(G)$. Thus, the validity check algorithm consists of a combination of inverse and forward NNs that, in addition to the inversion, computes a quality measure for the inversion:

$$\delta = ||S - F_{NN}(f_{NN}(S))|| \tag{9.12}$$

In conclusion, the solution to the problem of a scope check is obtained by estimating δ (9.12) where S is the result of the satellite measurement. This procedure (i) allows the detection of situations where the forward model or/and transfer function is inappropriate, (ii) does an "in scope" check for the retrieved parameters even if the domain has a complicated geometry, and (iii) can be adapted to all cases where a NN is used to emulate the inverse of an existing forward model.

9.6 Neural Network Emulations for SSM/I Data

In previous sections, we discussed the theoretical possibilities and premises for using NNs for modeling TFs and FMs. In this section, we illustrate these theoretical considerations using real-life applications of the NN approach to the SSM/I forward and retrieval problems. SSM/I is a well-established instrument, flown since 1987. Many different retrieval algorithms and several forward models have been developed for this sensor and several different databases are available for algorithm development and validation. Various different techniques have been applied to the algorithm development. Therefore, we can present an extensive comparison of different methods and approaches for this instrument. A raw buoy-SSM/I matchup database created by the Navy was used for the NN algorithm development, validation, and comparison. This database is quite representative, with the exception of high latitude and high wind speed events. In order to improve this situation the data sets were enriched by adding matchup databases collected by the high latitude European ocean weather ships Mike and Lima to the Navy database. Various filters have been applied to remove errors and noisy data (for a detailed discussion see Krasnopolsky 1997, and Krasnopolsky et al. 1999).

9.6.1 NN Emulation of the Empirical FM for SSM/I

The empirical SSM/I FM represents the relationship between the vector of geophysical parameters G and vector of satellite brightness temperatures (BTs) S, where $S = \{T19V,\ T19H,\ T22V,\ T37V,\ T37H\}$ ($TXXY$ means XX frequency in GHz and Y polarization). Four geophysical parameters are included in G (surface wind speed W, columnar water vapor V, columnar liquid water L, and SST or T_s). These

Table 9.1 Comparison of physically based radiative transfer and empirical NN forward models under clear and clear + cloudy (in parentheses) weather conditions

Author	Type	Inputs	BT RMS Error (K) Vertical	Horizontal
P & K (1992)	PB	*W, V, L, SST, Theta*[1], *P*0[2], *HWV*[3], *ZCLD*[4], *Ta*[5], *G*[6]	1.9 (2.3)	3.3 (4.3)
Wentz (1997)	PB	*W, V, L, SST, Theta*[1]	2.3 (2.8)	3.4 (5.1)
Krasnopolsky (1997)	NN	*W, V, L, SST*	1.5 (1.7)	3.0 (3.4)

[1] *Theta* – incidence angle
[2] *P*0 – surface pressure
[3] *HWV* – vapor scale height
[4] *ZCLD* – cloud height
[5] *Ta* – effective surface temperature
[6] *G* – lapse rate

are the main parameters influencing BTs measured by satellite sensors, which were used as inputs in the physically based FMs of Petty and Katsaros (1992, 1994) (referenced to below as PK) and Wentz (1997) (see Table 9.1). The NN emulation FM1 (Krasnopolsky 1997), which implements this SSM/I FM has four inputs $\{W,\ V,\ L,\ SST\}$, one hidden layer with 12 neurons, and five nonlinear BT outputs $\{T19V,\ T19H,\ T22V,\ T37V,\ T37H\}$ (see Fig. 9.2). The derivatives of the outputs with respect to the inputs, which can be easily calculated, constitute the Jacobian matrix $\boldsymbol{K}[S] = \{\partial S_i/\partial G_j\}$, which is required in the process of direct assimilation of the SSM/I BTs when the gradient of the SSM/I contribution to the cost function (9.11) χ_s is calculated

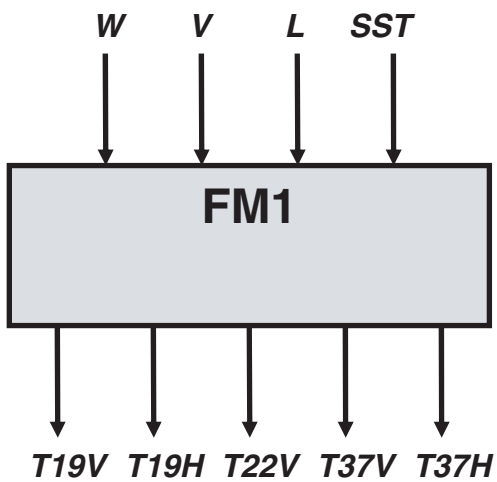


Fig. 9.2 SSM/I NN forward model, FM1 that generates brightness temperatures $S = TXXY(XX-$ frequency in GHz, Y – polarization) if given the vector G of four geophysical parameters: ocean surface wind speed (W), water vapor (V), liquid water (L) concentrations, and sea surface temperature (SST)

(Parrish and Derber 1992; Phalippou 1996). Estimating an NN emulation of FM and its derivatives is a much simpler and faster task than calculating radiative transfer forward models. The quality of the Jacobian matrix was evaluated in (Krasnopolsky 1997).

The matchup databases for the F11 SSM/I have been used for training (about 3,500 matchups) and validating (about 3,500 matchups) our forward model. FM1, the NN emulation of FM, was trained using all matchups that correspond to clear and cloudy weather conditions in accordance with the retrieval flags introduced by Stogryn et al. (1994). Only those cases where the microwave radiation cannot penetrate the clouds were removed. Then, more than 6,000 matchups for the F10 instrument were used for the testing and comparison of the FM1 with physically based forward models by PK and Wentz (1997). The RMS errors for FM1 are systematically better than those for the PK and Wentz FMs for all weather conditions and all channels considered. With the FM1, the horizontally polarized channels 19H and 37H have the highest RMSE, ~3.5 K under clear and ~4. K under clear and cloudy conditions. For the vertically polarized channels RMSEs are lower, 1.5 K under clear and 1.7 K under partly clear and partly cloudy conditions. The same trend can be observed for the PK and Wentz FMs. Table 9.1 presents total statistics (RMS errors) for the three FMs discussed here. RMS errors are averaged over different frequencies separately for the vertical and horizontal polarizations. RMS errors are higher under cloudy conditions because the complexity of the forward model increases due to the interaction of the microwave radiation with clouds.

Thus, FM1 gives results which are comparable or better in terms of RMSEs than the results

obtained with more sophisticated physically-based models (shown in Table 9.1), and is much simpler than physically based FMs. FM1 is not as general as a radiative transfer model; it was developed to be applied to the data assimilation system for variational retrieval and direct assimilation of SSM/I BTs at particular frequencies from a particular instrument. However, for this particular application (direct assimilation) and particular instrument it has a significant advantage (it is significantly simpler and faster), especially in an operational environment. This is also one of the applications where the accuracy of the NN Jacobian is essential. FM1 simultaneously calculates the BTs and Jacobian matrix. Krasnopolsky (1997) has demonstrated that for this particular application the NN Jacobian is sufficiently smooth. A generic NN ensemble technique is discussed by Krasnopolsky (2007) that improves the stability and reduces uncertainties of the NN emulation Jacobian if desired.

9.6.2 NN Empirical SSM/I Retrieval Algorithms

The SSM/I wind speed retrieval problem is a perfect example illustrating the general discussion presented in Section 9.2.1. The problems encountered in the case of SSM/I wind speed retrievals are very representative, and the methods used to solve them can easily be generalized for other geophysical parameters and sensors. About 10 different SSM/I wind speed retrieval algorithms, both empirical and physically-based, have been developed using a large variety of approaches and methods. Here these algorithms are compared in order to illustrate some properties of the different approaches mentioned in previous sections, and some advantages of the NN approach.

Goodberlet et al. (1989) developed the first global SSM/I wind speed retrieval algorithm. This algorithm is a single-parameter algorithm (it retrieves only wind speed) and is linear with respect to BTs (a linear multiple regression is used). Statistics for this algorithm are shown in Table 9.2 under the abbreviation GSW. This algorithm presents a linear approximation of a nonlinear (especially under cloudy sky conditions) SSM/I TF (9.8). Under clear sky conditions (Table 9.2), it retrieves the wind speed with an acceptable accuracy. However, under cloudy conditions where the amount of the water vapor and/or cloud liquid water in the atmosphere increases, errors in the retrieved wind speed increase significantly.

Goodberlet and Swift (1992) tried to improve the GSW algorithm performance under cloudy conditions, using nonlinear regression with a rational type of nonlinearity. Since the nature of the nonlinearity of the SSM/I TF under cloudy conditions is not known precisely, the application of such a nonlinear regression with a particular fixed type of nonlinearity may not be enough to improve results, as happens with the algorithm we refer to as GS. In many cases the GS algorithm generates false high wind speeds when real wind speeds are less than 15 m/s (Krasnopolsky et al. 1996).

A nonlinear (with respect to BTs) algorithm (called the GSWP algorithm here) introduced by Petty (1993) is based on a generalized linear regression. It presents a case where a nonlinearity introduced in the algorithm represents the nonlinear behavior of the TF much better. This algorithm introduces a nonlinear correction

Table 9.2 Errors (in m/s) for different SSM/I wind speed algorithms under clear and clear + cloudy (in parentheses) conditions

Algorithm	Method	Bias	Total RMSE	W > 15 m/s RMSE
GSW[1]	Multiple linear regression	−0.2 (−0.5)	1.8 (2.1)	(2.7)
GSWP[2]	Generalized linear regression	−0.1 (−0.3)	1.7 (1.9)	(2.6)
GS[3]	Nonlinear regression	0.5 (0.7)	1.8 (2.5)	(2.7)
Wentz[4]	Physically-based	0.1 (−0.1)	1.7 (2.1)	(2.6)
NN1[5]	Neural network	−0.1 (−0.2)	1.5 (1.7)	(2.3)
NN2[6]	Neural network	(−0.3)	(1.5)	–

[1]Goodberlet et al. (1989)
[2]Petty (1993)
[3]Goodberlet and Swift (1992)
[4]Wentz (1997)
[5]Krasnopolsky et al. (1996, 1999)
[6]Meng et al. (2007)

for the linear GSW algorithm when the amount of water vapor in the atmosphere is not zero. Table 9.2 shows that the GSWP algorithm improves the accuracy of retrievals compared to the linear GSW algorithm under both clear and cloudy conditions. However, it does not improve the GSW algorithm performance at high wind speeds because most of high wind speed events occur at mid- and high-latitudes where the amount of water vapor in the atmosphere is not significant. Here, the cloud liquid water is the main source of the nonlinear behavior of the TF and has to be taken into account.

NN algorithms have been introduced as an alternative to nonlinear and generalized linear regressions because the NN can model the nonlinear behavior of a TF better than these regressions. Stogryn et al. (1994) developed the first NN SSM/I wind speed algorithm, which consists of two NNs, each with the surface wind speed as a single output. One performs retrievals under clear and the other under cloudy conditions. Krasnopolsky et al. (1995) showed that a single NN with the same architecture (a single output) can generate retrievals for surface winds with the same accuracy as the two NNs developed by Stogryn et al. (1994) under both clear and cloudy conditions. Application of a single NN emulation led to a significant improvement in wind speed retrieval accuracy under clear conditions. Under higher moisture/cloudy conditions, the improvement was even greater (25–30%) compared to the GSW algorithm. The increase in areal coverage due to the improvements in accuracy was about 15% on average and higher in areas where there were significant weather events (higher levels of atmospheric moisture).

Both described above NN algorithms give very similar results because they had been developed using the same matchup database. This database, however, does not contain matchups for wind speed higher than about 20 m/s and contains very few matchups for wind speeds higher than 15 m/s. These algorithms are also single-parameter algorithms, i.e., they retrieve only the one parameter of wind speed; therefore, they cannot account for the variability in all related atmospheric (e.g., water vapor and liquid water) and surface (e.g., SST) parameters (which is especially important at higher wind speeds). This is why these NN algorithms pose the same problem; they cannot generate acceptable wind speeds at ranges higher then 18–19 m/s.

The next generation NN algorithm – a multi-parameter NN algorithm developed at NCEP (NN1 in Table 9.2) by Krasnopolsky et al. (1996, 1999) solved the high wind speed problem through three main advances. First, a new buoy/SSM/I matchup database was used in the development of this algorithm. It contained an extensive matchup data set for the F8, F10, and F11 sensors, provided by Navy Research Laboratory, and augmented with additional data from the European Ocean Weather Ships Mike and Lima for high latitude, high wind speed events (up to 26 m/s). Second, the NN training method was improved by enhancing the learning for the high wind speed range by weighting the high wind speed events. Third, the variability of related atmospheric and surface parameters was taken into account; surface wind speed (W), columnar water vapor (V), columnar liquid water (L), and SST are all retrieved simultaneously. In this case, the output vector of geophysical parameters is presented by $G = \{W,\ V,\ L,\ SST\}$. The NN1 algorithm uses five SSM/I channels, including 19 and 37 GHz for horizontal and vertical polarization and 22 GHz for vertical polarization (see Fig. 9.3).

Meng et al. (2007) (NN2 in Table 9.2) use the NN multi-parameter retrieval approach developed by Krasnopolsky et al. (1996, 1999) to design another NN multi-parameter retrieval algorithm for SSM/I. They

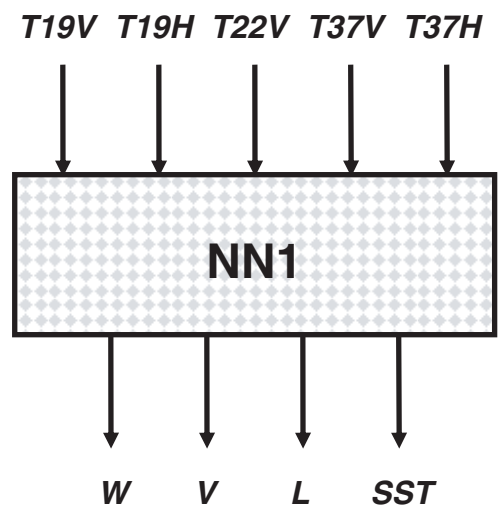


Fig. 9.3 SSM/I retrieval algorithm (NN1) emulating the inverse model to retrieve vector G of four geophysical parameters: ocean surface wind speed (W), water vapor (V), liquid water (L) concentrations, and sea surface temperature (SST) if given five brightness temperatures $S = TXXY$ (XX – frequency in GHz, Y – polarization)

use all 7 SSM/I BTs as inputs. Their output vector also has four components $G = \{W, T_a, H, SST\}$ where surface wind speed (W), surface air temperature (T_a), humidity (H), and *SST* are retrieved simultaneously. In this case, the training database was limited by maximum wind speeds of about 20 m/s. Moreover, there are only a few higher speed events with $W > 15$–17 m/s.

Table 9.2 shows a comparison of the performance of all the aforementioned empirical algorithms in terms of the accuracy of the surface wind speed retrievals. It also shows statistics for a physically based algorithm developed by Wentz (1997), which is based on a linearized numerical inversion (9.7) of a physically based FM. The statistics presented in Table 9.2 were calculated using independent buoy-SSM/I matchups. Table 9.2 shows that the NN algorithms outperform all other algorithms. All algorithms except the NN algorithms show a tendency to overestimate high wind speeds. This happens because high wind speed events are usually accompanied by a significant amount of cloud liquid water in the atmosphere. Under these circumstances the transfer function f becomes a complicated nonlinear function, and simple one-parametric regression algorithms cannot provide an adequate representation of this function and confuse a high concentration of cloud liquid water with very high wind speeds. Krasnopolsky et al. (1999, 2000) showed that single-parameter algorithms have additional (compared to multi-parameter retrievals) systematic (bias) and random (unaccounted variance due to other parameters) errors in a single retrieved parameter because of described effects. NN1 shows the best total performance, in terms of bias, RMSE, and high wind speed performance.

As mentioned above, one of the significant advantages of NN1 algorithm is its ability to retrieve simultaneously not only the wind speed but also the three other atmospheric and ocean surface parameters columnar water vapor V, columnar liquid water L, and *SST*. Krasnopolsky et al. (1996) showed that the accuracy of retrieval for other geophysical parameters is very good and close to those attained by the algorithms of the Alishouse et al. (1990) (for V) and Weng and Grody (1994) (for L). In addition, Krasnopolsky et al. (1999, 2000) have shown that the errors of multi-parameter NN algorithms have a weaker dependence on the related atmospheric and surface parameters than the errors of the single-parameter algorithms considered. The retrieved *SST* in this case is not accurate (the RMS error is about 4°C, see Krasnopolsky et al. 1996); however, including *SST* in the vector of retrieved parameters decreases the error in other retrievals correlated with the *SST*. For the multi-parameter NN algorithm NN2 (Meng et al. 2007), the choice of the additional outputs surface air temperature (T_a) and humidity (H), that are closely and physically related and correlated with *SST*, makes the accuracy of the retrieved SST signal higher (the bias is about 0.1°C and RMS error 1.54°C). In accordance with the classical, "linear" remote sensing paradigm, the SSM/I instrument does not have the frequency required to sense SST. However, due to the nonlinear nature of the NN emulation and the proper choice of output parameters the multi-parameter NN algorithm is probably able to use weak nonlinear dependencies between NN inputs and outputs and between NN outputs to retrieve SST with a high accuracy.

9.6.3 Controlling the NN Generalization in the SSM/I Case

The NN1 retrieval algorithm was judged so successful that it has been used as the operational algorithm in the global data assimilation system at NCEP/NOAA since 1998. Given five brightness temperatures, it retrieves the four geophysical parameters ocean surface wind speed, water vapor and liquid water concentrations, and sea surface temperature. At high levels of liquid water concentration the microwave radiation cannot penetrate clouds and surface wind speed retrievals become impossible. Brightness temperatures on these occasions fall far outside the training domain D_T. The retrieval algorithm in these cases, if not flagged properly, will produce wind speed retrievals which are physically meaningless (i.e., not related to actual surface wind speed). Usually a statistically-based retrieval flag, based on global statistics, is used to indicate such occurrences. Under complicated local conditions, however, it can produce significant number of false alarms, or does not produce alarms when needed.

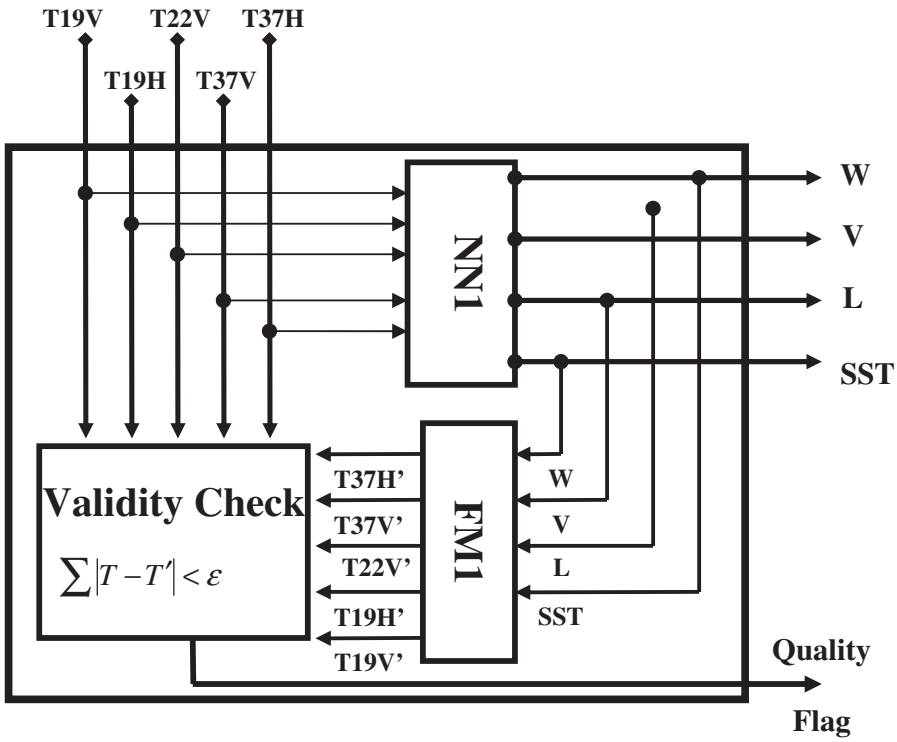


Fig. 9.4 SSM/I retrieval algorithm (NN1) emulating the inverse model to retrieve vector G of four geophysical parameters: ocean surface wind speed (W), water vapor (V) and liquid water (L) concentrations, and sea surface temperature (SST) if given five brightness temperatures $S = TXXY$ (XX – frequency in GHz, Y – polarization). This vector G is fed to the FM1 emulating the forward model to get brightness temperatures $S' = TXXY'$. The difference $\Delta S = |S - S'|$ is monitored and raises a warning flag if it is above a suitably chosen threshold

The validity check shown in Fig. 9.4, if added to a standard retrieval flag, helps indicate such occurrences. The NN SSM/I forward model FM1 is used in combination with the NN1 retrieval algorithm. For each satellite measurement S, the geophysical parameters retrieved from brightness temperatures S are fed into the NN SSM/I forward model which produces another set of brightness temperatures S'. For S within the training domain ($S \in D_T$) the difference, $\Delta S = |S - S'|$, is sufficiently small. For S outside the training domain the larger difference raises a warning flag, if it is above a suitably chosen threshold. Krasnopolsky and Schiller (2003) showed the percentage of removed data and improvement in the accuracy of the wind speed retrievals as functions of this threshold. They showed that applying the generalization control reduces the RMS error significantly; the maximum error is reduced even more. This means that this approach is very efficient for removing outliers.

9.7 Discussion

In this chapter we discussed a broad class of NN applications dealing with the solution of the RS forward and inverse problems. These applications are closely related to the standard and variational retrievals, which estimate geophysical parameters from remote satellite measurements. Both standard and variational techniques require a data converter to convert satellite measurements into geophysical parameters or vice versa. Standard retrievals use a TF (a solution of the inverse problem) and variational retrievals use a FM (a solution of the forward problem) for this purpose. In many cases the TF and the FM can be considered as continuous nonlinear mappings. Because the NN technique is a general technique for continuous nonlinear mapping, it can be used successfully for modeling TFs and FMs.

Theoretical considerations presented in this section were illustrated using several real-life applications that

exemplify a NN based intelligent integral approach (e.g., the approach and design presented in Fig. 9.4) where the entire retrieval system, including the quality control block, is designed from a combination of several specialized NNs. This approach offers significant advantages in real life operational applications. This intelligent retrieval system produces not only accurate retrievals but also performs an analysis and quality control of the retrievals and environmental conditions, rejecting any poor retrievals that occur.

The NN applications presented in this section illustrate the strengths and limits of the NN technique for inferring geophysical parameters from remote sensing measurements. The success of the NN approach, as any nonlinear statistical approach, strongly depends on the adequacy of the data set used for the NN training. The data availability, precision, quality, representativeness, and amount are crucial for success in this type of NN application. However, NNs successfully compete with other statistical methods and usually perform better because they are able to emulate the functional relationship between inputs and the outputs in an optimal way. NNs can successfully compete with even physically based approaches because, in many cases, explicit knowledge of very complicated physical processes in the environment is limited and a NN based empirical approach is more appropriate. It can take into account more physics implicitly using the NN ability to learn from data inherent dependencies, than a physically based approach would include explicitly.

References

Abdelgadir, A. et al. (1998). Forward and inverse modeling of canopy directional reflectance using a neural network. *International Journal of Remote Sensing*, *19*, 453–471.

Aires, F., Rossow, W. B., Scott, N. A, & Chedin, A. (2002). Remote sensing from the infrared atmospheric sounding interferometer instrument. 2. Simultaneous retrieval of temperature, water vapor, and ozone atmospheric profiles. *Journal of Geophysical Research*, *107*, 4620.

Aires, F., Prigent, C., & Rossow, W. B. (2004). Neural network uncertainty assessment using Bayesian statistics with application to remote sensing. 3. Network Jacobians. *Journal of Geophysical Research*, *109*, D10305.

Alishouse, J. C. et al. (1990). Determination of oceanic total precipitable water from the SSM/I. *IEEE Transactions on Geoscience and Remote Sensing*, *GE-23*, 811–816.

Atkinson, P. M., & Tatnall, A. R. L. (1997). Neural networks in remote sensing – introduction. *International Journal of Remote Sensing*, *18*(4), 699–709.

Attali, J.-G., & Pagès, G. (1997). Approximations of functions by a multilayer perceptron: A new approach. *Neural Networks*, *10*, 1069–1081.

Cabrera-Mercader, C. R., & Staelin, D. H. (1995). Passive microwave relative humidity retrievals using feedforward neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, *33*, 1324–1328.

Chen, T., & Chen, H. (1995a). Approximation capability to functions of several variables, nonlinear functionals and operators by radial basis function neural networks. *Neural Networks*, ***6***, 904–910.

Chen, T., & Chen, H. (1995b). Universal approximation to nonlinear operators by neural networks with arbitrary activation function and its application to dynamical systems. *Neural Networks*, *6*, 911–917.

Cornford, D., Nabney, I. T., & Ramage, G. (2001). Improved neural network scatterometer forward models. *Journal of Geophysical Research*, *106*, 22331–22338.

Cybenko, G. (1989). Approximation by superposition of sigmoidal functions. *Mathematics of Control Signals and Systems*, *2*, 303–314.

Davis, D. T. et al. (1995). Solving inverse problems by Bayesian iterative inversion of a forward model with application to parameter mapping using SMMR remote sensing data. *IEEE Transactions on Geoscience and Remote Sensing*,33, 1182–1193.

Derber, J. C., & Wu, W.-S. (1998). The use of TOVS cloud-cleared radiances in the NCEP SSI analysis system. *Monthly Weather Reviews*, *126*, 2287–2299.

DeVore, R. A. (1998). Nonlinear approximation. *Acta Numerica*, *8*, 51–150.

Eyre, J. R., & Lorenc, A. C. (1989). Direct use of satellite sounding radiances in numerical weather prediction. *Meteorology Magazine*, *118*, 13–16.

Funahashi, K. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*, *2*, 183–192.

Goodberlet, M. A., & Swift, C. T. (1992). Improved retrievals from the DMSP wind speed algorithm under adverse weather conditions. *IEEE Transactions on Geoscience and Remote Sensing*, ***30***, 1076–1077.

Goodberlet, M. A., Swift, C. T., & Wilkerson, J. C. (1989). Remote sensing of ocean surface winds with the special sensor microwave imager *Journal of Geophysical Research*, *94*, 14547–14555.

Hadamard, J. (1902). Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton University Bulletin*, *13*, 49–52.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward network. *Neural Networks*, *4*, 251–257.

Krasnopolsky, V. (1997). A neural network-based forward model for direct assimilation of SSM/I brightness temperatures. *Technical note* (OMB contribution No. 140). NCEP/NOAA, Camp Springs, MD 20746.

Krasnopolsky, V., Breaker, L. C., & Gemmill, W. H. (1995). A neural network as a nonlinear transfer function model for retrieving surface wind speeds from the special sensor microwave imager. *Journal of Geophysical Research.*, *100*, 11033–11045.

Krasnopolsky, V., Gemmill, W. H., & Breaker, L. C. (1996). A new transfer function for SSM/I based on an expanded neural

network architecture. *Technical note* (OMB contribution No. 137). NCEP/NOAA, Camp Springs, MD 20746.

Krasnopolsky, V. M. (2007). Reducing uncertainties in neural network Jacobians and improving accuracy of neural network emulations with NN ensemble approaches. *Neural Networks*, *20*, 454–461.

Krasnopolsky, V. M., & Schiller, H. (2003). Some neural network applications in environmental sciences. Part I: Forward and inverse problems in satellite remote sensing. *Neural Networks*, *16*, 321–334.

Krasnopolsky, V. M., Gemmill, W. H., & Breaker, L. C. (1999). A multiparameter empirical ocean algorithm for SSM/I retrievals. *Canadian Journal of Remote Sensing*, *25*, 486–503.

Krasnopolsky, V. M., Gemmill, W. H., & Breaker, L. C. (2000). A neural network multi-parameter algorithm SSM/I ocean retrievals: Comparisons and validations. *Remote Sensing of Environment*, *73*, 133–142.

Lorenc, A. C. (1986). Analysis methods for numerical weather prediction. *Quarterly Journal of Royal Meteorology Society*, *122*, 1177–1194.

McNally, A. P., Derber, J. C., Wu, W.-S., & Katz, B. B. (2000). The use of TOVS level 1B radiances in the NCEP SSI analysis system. *Quarterly Journal of Royal Meteorological Society*, *126*, 689–724.

Meng, L. et al. (2007). Neural network retrieval of ocean surface parameters from SSM/I data. *Monthly Weather Review*, *126*, 586–597.

Mueller, M. D. et al. (2003). Ozone profile retrieval from global ozone monitoring experiment data using a neural network approach (Neural Network Ozone Retrieval System (NNORSY)). *Journal of Geophysical Research*, *108*, 4497.

Parker, R. L. (1994). *Geophysical inverse theory* (400 pp.). Princeton, NJ: Princeton University Press.

Parrish, D. F., & Derber, J. C. (1992). The national meteorological center's spectral statistical-interpolation analysis system. *Monthly Weather Review*, *120*, 1747–1763.

Petty, G. W. (1993). A comparison of SSM/I algorithms for the estimation of surface wind. *Proceedings of Shared Processing Network DMSP SSM/I Algorithm Symposium*, Monterrey, CA, June 8–10, 1993.

Petty, G. W., & Katsaros, K. B. (1992). The response of the special sensor microwave/imager to the marine environment. Part I: An analytic model for the atmospheric component of observed brightness temperature. *Journal of Atmospheric Oceanic Technology*, *9*, 746–761.

Petty, G. W., & Katsaros, K. B. (1994). The response of the SSM/I to the marine environment. Part II: A parameterization of the effect of the sea surface slope distribution on emission and reflection. *Journal of Atmospheric Oceanic Technology*, *11*, 617–628.

Phalippou, L. (1996). Variational retrieval of humidity profile, wind speed and cloud liquid–water path with the SSM/I: Potential for numerical weather prediction. *Quarterly Journal of Royal Meteorological Society*, *122*, 327–355.

Pierce, L., Sarabandi, K., & Ulaby, F. T. (1994). Application of an artificial neural network in canopy scattering inversion. *International Journal of Remote Sensing*, *15*, 3263–3270.

Prigent, C., Phalippou, L., & English, S. (1997). Variational inversion of the SSM/I observations during the ASTEX campaign. *Journal of Applied Meteorology*, *36*, 493–508.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, & P. R. Group (Eds.), *Parallel distributed processing* (Vol. 1, pp. 318–362). Cambridge, MA: MIT Press.

Schiller, H., & Doerffer, R. (1999). Neural network for emulation of an inverse model - operational derivation of case II water properties from MERIS data. *International Journal of Remote Sensing*, *20*, 1735–1746.

Schiller, H., & Krasnopolsky, V. M. (2001). Domain check for input to NN emulating an inverse model. *Proceedings of International Joint Conference on Neural Networks*, Washington, DC, July 15–19, pp. 2150–2152.

Smith, J. A. (1993). LAI inversion using a back-propagation neural network trained with a multiple scattering model. *IEEE Transactions on Geoscience and Remote Sensing*, *GE-31*, 1102–1106.

Stoffelen, A., & Anderson, D. (1997). Scatterometer data interpretation: Estimation and validation of the transfer function CMOD4. *Journal of Geophysical Research*, *102*, 5767–5780.

Stogryn, A. P., Butler, C. T., & Bartolac, T. J. (1994). Ocean surface wind retrievals from special sensor microwave imager data with neural networks. *Journal of Geophysical Research*, *90*, 981–984.

Thiria, S., Mejia, C., Badran, F., & Crepon, M. (1993). A neural network approach for modeling nonlinear transfer functions: Application for wind retrieval from spaceborn scatterometer data. *Journal of Geophysical Research*, *98*, 22827–22841.

Tsang, L. et al. (1992). Inversion of snow parameters from passive microwave remote sensing measurements by a neural network trained with a multiple scattering model. *IEEE Transactions on Geoscience and Remote Sensing*, *GE-30*, 1015–1024.

Vapnik, V. N. (1995). *The nature of statistical learning theory* (189 pp.). New York: Springer.

Vapnik, V. N., & Kotz, S. (2006). *Estimation of dependences based on empirical data (Information Science and Statistics)* (495 pp.). New York: Springer.

Weng, F., & Grody, N. G. (1994). Retrieval of cloud liquid water using the special sensor microwave imager (SSM/I). *Journal of Geophysical Research*, *99*, 25535–25551.

Wentz, F. J. (1997). A well-calibrated ocean algorithm for special sensor microwave/imager. *Journal of Geophysical Research*, *102*, 8703–8718.

Implementing a Neural Network Emulation of a Satellite Retrieval Algorithm

10

George S. Young

10.1 Introduction

As shown in Stogryn et al. (1994), Krasnopolsky et al. (1995), Thiria et al. (1993), Cornford et al. (2001), and many other studies summarized in Chapter 9, neural networks (NN) can be used to emulate the physically-based retrieval algorithms traditionally used to estimate geophysical parameters from satellite measurements. The tradeoff involved is a minor sacrifice in accuracy for a major gain in speed, an important factor in operational data analysis. This chapter will cover the design and development of such networks, illustrating the process by means of an extended example. The focus will be on the practical issues of network design and troubleshooting. Two topics in particular are of concern to the NN developer: computational complexity and performance shortfalls. This chapter will explore how to determine the computational complexity required for solving a particular problem, how to determine if the network design being validated supports that degree of complexity, and how to catch and correct problems in the network design and developmental data set.

As discussed in Chapter 9, geophysical remote sensing satellites measure either radiances using passive radiometers or backscatter using a transmitter/receiver pair. The challenge is then to estimate the geophysical parameters of interest from these measured quantities. The physics-based forward problem (equation 9.4) captures the cause and effect relationship between the geophysical parameters and the satellite-measured quantities. Thus, the forward problem must be a single-valued function (i.e. have a single possible output value for each set of input values) if we have access to all of its input parameters. As a result, the forward problem is generally well-posed, i.e. variations in the input parameters are not grossly amplified in the output. One could, however, imagine some geophysical processes for which the forward problem was ill-posed for some parameter values as a result of a sudden transition from one regime of behavior to another (e.g. the onset of fog formation producing a sharp change in shortwave albedo in response to a minor change in air temperature).

In contrast to the forward problem, the inverse problem of deducing geophysical parameters from satellite measurements does not follow cause and effect and so is not necessarily single-valued over the bulk of the input parameter space. Thus, these satellite retrieval problems are often ill-posed over a significant part of their input parameter space. As a result, the transfer function (equation 9.5) can be multi-valued, having a finite number of solutions at each point in some regions ofthe input parameter space and an infinite number of solutions at points along the borders of these regions. These issues must be addressed via some form of regularization (i.e. using prior knowledge of the problem's physics to constrain the solution in some way). The extended example described below exhibits all of these problems.

The example examined here is the classic problem of retrieving sea surface wind speed from satellite measurements of sea surface backscatter. In this case, a Synthetic Aperture Radar (SAR) is used to measure the backscatter from short-wavelength, wind-driven

George S. Young (✉)
Department of Meteorology, The Pennsylvania State University, PA, USA
Address: 503 Walker Building, University Park, PA 16802, USA
Phone: 814-863-4228; fax (814) 865-9429;
email: young@meteo.psu.edu

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

ocean waves. Because the wave amplitude is driven by the wind-induced surface stress, the forward or causal direction is from wind speed to backscatter. Thus, the problem of interest for remote sensing is the inverse or retrieval problem.

The traditional approach to solving the SAR wind speed retrieval problem makes use of a semi-empirical model that exists for the forward problem (Monaldo et al. 2001). The version used here is called CMOD4 (Stoffelen and Anderson 1997a, b). This forward model is, however, too complex to invert analytically, so that approach can't be used to produce an analytic model for the inverse problem. Instead, the traditional approach is to run the forward problem for the full range of expected wind speeds, and then select the speed for which it predicts a backscatter value closest to that observed. This approach involves running the computationally expensive physics-based forward model many (typically 25) times for each pixel in a SAR image, thus restricting realtime applications and bulk data analysis.

NN, however, are well suited to emulating smooth nonlinear functions such as that in the forward (CMOD4) model (see Chapter 9). This raises a number of questions concerning their potential for improving the efficiency of the wind speed retrieval process. Could we achieve speed up (i.e. use fewer or faster floating point operations and intrinsic function calls) by using a NN to emulate the forward model? Could we go one step further and use a NN to emulate the inverse model even though its analytic form is not known? Because we have a physics-based analytic model for the forward problem we can generate the data required to train a NN for either the forward or inverse problem as discussed in Chapter 9.

There are, thus, several ways in which NN could be applied to solve the SAR wind speed retrieval problem. First, one can emulate the forward model with a NN and retain the current brute-force-search approach (i.e. examining each of the 25 candidates) to finding a wind speed that yields the observed backscatter. While likely to succeed because the forward physics is generally well-posed, this approach maintains the fundamental inefficiency of the traditional method, using many forward model runs to obtain the answer for each pixel. Second, one can emulate the inverse model with a NN. This is a potentially more challenging problem because the physics is apt to be ill-posed in at least part of the input parameter space. It does, however, offer a distinct advantage in terms of efficiency because only one model run is required per pixel. We'll explore the pros and cons of these two approaches as the example unfolds below. Third, one can follow the approach suggested by Thiria et al. (1993) for similar problems in scatterometry, emulating the inverse model with a categorization rather than regression NN. This approach only partially mitigates the inefficiency of using a multi-run forward model to solve the retrieval problem because, although the NN is run only once, the network's output layer has a node for each of the discretized values of wind speed. The activation output of each of these output nodes is the coefficient of likelihood for the corresponding value of wind speed. A continuous value of wind speed is obtained by fitting an interpolation curve to the peak of this likelihood function. The Thiria et al. method does, however, offer the interesting benefit of documenting whether the inverse function is single- or multi-valued: a single peak will occur in the plot of likelihood coefficient versus wind speed if inverse function is single-valued and multiple peaks will occur if it is multi-valued. Thus, this third approach can serve a valuable diagnostic role, guiding the development of more efficient methods based on the inverse regression model approach.

10.2 Method

The success of a NN development project depends crucially on two factors, use of an appropriate training data set and design of a minimally complex, but sufficient network. As we shall see as the example progresses, both of these factors can pose serious challenges in ill-posed retrieval problems.

10.2.1 Developmental Data

As pointed out in Chapter 9, the existence of an analytic forward model such as CMOD4 offers several advantages when developing a NN for the corresponding retrieval problem. Using the forward model to generate the training datasets allows us to cover the entire parameter space with cases, including types of events that would be rare in observations. This eliminates one of the common problems encountered when training a NN from observations wherein the observation set,

and thus the network's fitness, is dominated by the climatologically most likely parts of the input parameter space. By using the physics-based forward model to uniformly cover all parts of the parameter space with cases, we can ensure that the NN is trained to perform as well in the rarer parts of the input parameter space as it does in the more commonly encountered cases. A NN trained in this manner functions in interpolation mode for all cases rather than having to extrapolate from the training set when dealing with the rarer events. As discussed in Chapter 9, this approach leads to better generalization, i.e. better performance in actual operations.

For the SAR wind speed problem the parameters are surface wind speed, backscatter, incidence angle, and the radar look direction relative to the surface wind direction (Monaldo et al. 2001). For the forward model all but backscatter are input parameters and for the inverse model all but wind speed are. Thus, to obtain developmental data for both the forward and inverse models, CMOD4 was used to create a uniform four-dimensional grid of backscatter values with cases spaced every 1° in incidence angle, every 1° in radar look angle relative to the wind direction, and at 450 values of wind speed from 0 to 60 ms^{-1}. This results in 5,670,000 cases, more than can be processed in a reasonable time via the NN training algorithm. Therefore two samples of 10,000 cases are selected at random, one for training the network, a second for validation, and a third for final validating. Given the independence of the samples, high performance on the validation set indicates that the NN has been fit robustly across the input parameter space. Of course, if the same validation set is used repeatedly when optimizing the NN architecture, the resulting NN will be biased towards it as it is towards the developmental data. Therefore, a third set of 10,000 cases is used to recalculate the skill of the final network, eliminating this source of false optimism. Alternatively, one could use a different validation set on each of the NN architectures in order to achieve this independence.

10.2.2 Neural Network Design

The basics of NN are discussed in chapter Y-NN and one approach to universal function emulation using NNs is explored in Chapter 9. Although details can vary from one implementation to another, the basics are the same. The network's input layer consists of a data source for each of the input parameters. All of these values are fed into each of the processing nodes of the next layer, often called a hidden layer. Each processing node first computes the output of a linear equation, which takes its input from the input layer. The coefficients can differ between processing nodes, so differing computations are done in parallel by the nodes of a processing layer. The results of each equation are then "squashed" into the range -1 to 1 using the hyperbolic tangent or a similar activation function. At this point the squashed results may be fed into an output layer as in Chapter 9 or into additional processing layers as in Reed and Marks (1999). The latter approach will be used in the example discussed in this chapter. In either case, the results of the final processing layer are fed through one more linear equation, called an output node. If the result is to be a probabilistic or categorical (yes/no) prediction, than a squashing function is applied to the output layer's result. If, as in the SAR wind speed retrieval problem, the desired output is numerical, no squashing function is applied to the output node.

Training of the NN involves finding values for all of these coefficients. This can be done via various forms of the back propagation of errors (Reed and Marks 1999) or by using a general purpose nonlinear optimization tool such a genetic algorithm (GA, Haupt and Haupt 2004) (Jones 1993; Munro 1993). Neither of these approaches is necessarily best for all retrieval problems. For example, training a NN to emulate some nonlinear functions is "GA-easy" while training the network to emulate other functions of equivalent complexity is "GA-hard". The "GA-hard" problems are those where the under-constrained nature of NNs leads to multiple distinct solutions, each at a local minimum in the network's forecast error, that, when bred, yield solutions of lower rather than higher skill (Reed and Marks 1999). Likewise networks trained by back propagation can become trapped in such local minima (Yamada and Yabuta 1993; Reed and Marks 1999). Back propagation techniques are popular, particularly in the data mining community, (e.g. Alpsan et al. 1995; Reed and Marks 1999; Witten and Frank 2005) and will be employed here.

As mentioned in Chapter 9, NNs with a single hidden layer are adequate to fit any continuous nonlinear function provided enough hidden nodes are used.

Thus, the design process for a single-layer network consists of simply increasing the number of nodes until good performance is achieved on independent validation data.[1] The smallest network that achieves good performance is then used in practice, both because it is more efficient than larger networks and because it is less likely to be over-fit and so will generalize better. The design problem is more complex when the network includes multiple processing layers. The designer not only has to decide on the number of layers, but also on the number of nodes per layer. The design space is thus multidimensional. One could, of course, use a genetic algorithm or some other general purpose nonlinear optimizer to develop the network design (Dodd 1990; Reed and Marks 1999), but in practice one often simply tries a number of likely candidates learning enough from each to design a likely successor.

While this iterative design process includes two relatively distinct aspects, determining how many layers to use and determining how many nodes to include in each, there is a tradeoff between the two because increasing the number of layers decreases the number of nodes required in each. The design tactic employed here is to alternately tune the number of nodes and the number of layers, starting first with the nodes and then cycling back and forth until performance stops improving. Experience suggests that three hidden layers makes a good starting point with the first layer having as many nodes as there are predictors and the other layers progressively less. The network designer then tunes the number of nodes per layer by trying greater and lower node counts in the networks and seeing which produces the best result. This is just the manual version of the various gradient descent methods common to nonlinear optimization (Nelder and Mead 1965). Once the optimal number of nodes is found, one should check the weights in the last hidden layer. If each node has the same weight values as all the others, then they're all making the same calculation and the layer is probably redundant. In some cases the results will improve by eliminating that layer and in other cases by reducing the number of nodes it contains until the weights differ from node to node. If the nodes in the last hidden layer differ, one could try adding an additional layer of two nodes. If performance improves, one could start tuning the number of nodes again, otherwise go back one design and stop iterating. If this approach seems too complex, stick with a single hidden layer as described in Chapter 9.

[1] In a more general setting, for example with noisy data, the optimal NN is the one that has the lowest average error over multiple validation sets.

10.2.3 Network Training

We could easily develop our own back propagation training program following, for example, Reed and Marks (1999). Ready built open source tools such as Weka (Witten and Frank 2005) are, however, readily available for download (http://www.cs.waikato.ac.nz/ml/weka/), so that approach is often more efficient. Weka will be used here because it offers good documentation, a point-and-click interface, and extensive support for troubleshooting and validation. No matter which development tool we use, the phases are the same.

- Data ingest
- Predictand and predictor selection
- Cross-validation setup (in this example using a validation set)
- Network design
- Network building (i.e. tuning the regression coefficients)
- Network testing on independent validation data

Following each validation stage the network is redesigned as described above and the build and validate cycle is repeated. Clearly, to make this cycle converge, careful consideration must be given to how the validation results vary with network design. The final design must yield a high accuracy solution but do so with the minimum number of nodes, thereby minimizing computational cost.

10.3 Results

The NN development tactics described above are applied to both the forward and inverse problems linking SAR backscatter with surface wind speed. The results will be presented in the order originally encountered when this project was under development. Each of the problems that arose during this development

process will be discussed at the stage where it was first diagnosed. Thus, the results discussion will focus on how the problems were detected and then addressed by appropriate regularization. The section will conclude with a discussion of efficiency issues and a comparison of the various NN approaches to the traditional CMOD4 approach they are designed to replace.

10.3.1 Forward Model

Table 10.1 shows a sample of network designs for the forward (wind speed to backscatter) problem. The network design is described via a comma separated list giving the number of nodes in each hidden layer. Training was undertaken for 500 cycles (Epochs) in all but one case. The model performance is described in terms of percent variance explained (i.e. r^2) and mean absolute error (MAE). Each network's computational cost is computed from a count of the various floating point operations and intrinsic function calls using times estimated for each of these actions via a procedure described in Section 10.3.4 below. The cost value is then normalized by the similarly computed run time estimate for the analytic CMOD4 code. Lower values of cost are of course to be preferred over high values. Note that these values are not the cost of network development, but rather the computational cost to apply the network in operational SAR image analysis.

The first NN, with a single hidden layer of four nodes shows skill, but is not accurate enough to serve as a replacement for the analytic CMOD4 forward model as the MAE remains large relative to operational requirements. Performance improves when two more hidden layers are added and further improves when the number of nodes in these layers is increased past a total of 20. There is, however, some decrease in skill for the most complex networks, suggesting either over-fitting or an inadequate number of training epochs for a network of that degree of network complexity. Because the skill improves markedly when the most complex network is redeveloped using 5,000 epochs instead of 500, we can conclude that inadequate training rather than over-fitting was the problem. This final NN was reevaluated on an independent set of test data yielding a percent variance explained of 0.9998, the same value obtained with the validation data set. Thus, we can conclude that the NN was not over-fit, a fact attributable to the noise free synthetic data and the large number of cases in the developmental data set. The MAE of 0.0052 is accurate enough that the NN could be used as a replacement for the analytic CMOD4 forward model. It is, however, a factor of five more costly in computational time than the analytic CMOD4 model. Clearly, if NNs are to be of aid in the SAR wind speed analysis it must be in the inverse rather than forward problem, where it would replace multiple runs of the analytic CMOD4 model instead of just one.

10.3.2 Inverse Model

Given the successful emulation of the analytic CMOD4 forward model, it is reasonable to ask whether the neural net approach can be applied to emulate the inverse problem for which no analytic model exists. Doing so would eliminate the need for multiple runs of the analytic forward model to make up for the lack of an analytic inverse model. As Table 10.2 shows, however, the initial NN results are quite disheartening.

The skill demonstrated by each NN was markedly worse on the inverse problem than it had been on the forward problem discussed in the section above.

Table 10.1 Neural net design and performance for the forward problem. Those values of r^2 and MAE in parentheses were computed using an independent set of test data, the rest were all computed using a single set of validation data

Network design	Training epochs	$\mathbf{r}^2$	MAE	Relative cost
4	500	0.9577	0.15	0.28
4,3,2	500	0.9616	0.10	0.63
8,4,2	500	0.9631	0.12	1.03
12,6,2	500	0.9956	0.050	1.55
16,8,4	500	0.9972	0.023	2.29
24,12,2	500	0.9984	0.015	3.98
32,16,2	500	0.9962	0.035	4.84
32,16,4	500	0.9884	0.043	5.06
32,16,4	5,000	0.9998 (0.9998)	0.0053 (0.0052)	5.06

Table 10.2 Neural net design and performance for the inverse problem. Those values of r^2 and MAE in parentheses were computed using an independent set of test data, the rest were all computed using a single set of validation data

Network design	Training epochs	r^2	MAE (ms^{-1})	Relative cost
12,6,3	500	0.8188	5.02	0.0912
16,8,4	500	0.8714	4.03	0.1281
24,16,2	500	0.8712	3.75	0.2195
32,16,2	500	0.8571	4.53	0.2713
32,16,4	500	0.8709	4.12	0.2832
32,16,4	5,000	0.8541 (0.8506)	3.80 (3.82)	0.2832

Indeed, the best r^2 value was near 0.87 rather than above 0.999. Another sign that the NN is not performing well is that further training on the inverse problem did not provide a performance improvement the way it did on the forward problem. Thus, something about the inverse problem is making it much more challenging for neural net emulation. The problem cannot be attributed to reuse of the validation data because similar skill was obtained for the most complex network when it was reevaluated using an independent set of test data.

Examination of the relationship between wind speed and backscatter in CMOD4 quickly reveals the source of the problem. For a number of incidence angle and wind direction combinations, such as that shown in Fig. 10.1, the relationship is multi-valued in wind speed. Thus, while each wind speed corresponds to one backscatter value, a single backscatter may correspond to two widely different wind speeds. Because the NN has only one output it can't fit a multi-valued function, and thus fails over much of its input parameter space.

There are two obvious approaches for solving this problem. One approach is to redesign the training set and the NNs to support two output values. These values would differ for those parts of the input parameter space where wind speed is a multi-valued function of backscatter, while being the same where the function is single-valued. This approach leaves the decision as to which value is correct to the end user, thus putting off the required regularization of the inverse problem. The second approach is to regularize the data in the training set as described in Chapter 9, so that the NN is developed to return only one of the two possible wind speed values. This approach most closely follows the

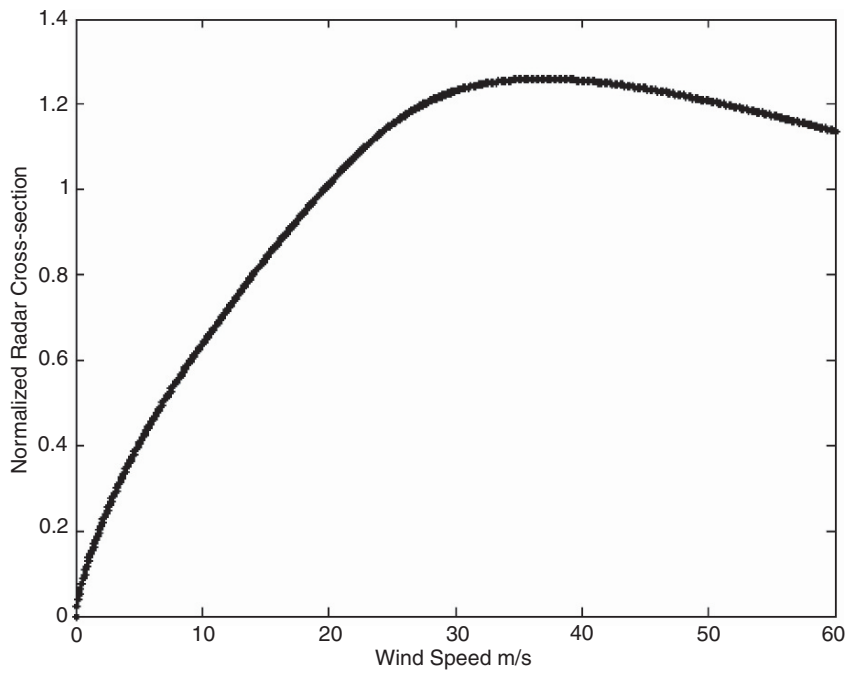


Fig. 10.1 Normalized radar cross-section (i.e. backscatter) as a function of wind speed for an incidence angle of 20° and a wind-relative look direction of 45°

current practice with the analytic CMOD4 model and so will be followed in the sections below.

10.3.3 Regularization of the Inverse Model

A simple regularization of the training data set can be achieved by eliminating those cases for which the wind speed takes on the higher of two possible values. This follows the current CMOD4 practice (N. S. Winstead 2006, personal communication) of assuming that the lowest wind speed consistent with the backscatter is most likely to be correct. This regularization makes the function to be emulated single-valued and thus should improve the NN's performance on the inverse problem.

As shown by comparing Tables 10.2 and 10.3, however, this improvement does not occur (except for the r^2 statistic for one network design). Rather, the percent variance explained dropped slightly and the MAE increased. Why did the regularization fail to make the problem more tractable? Reexamination of Fig. 10.1 sheds light on the problem. Not only is wind speed potentially a multi-valued function of backscatter, but also it exhibits a zone of hypersensitivity to backscatter near the peak of this latter parameter. Thus, for backscatter observations near this peak value, major changes in wind speed result from minor changes in backscatter. So backscatter provides little information on wind speed in that zone. This sensitivity makes the NN hard to train and would in any case cause the resulting network's output to be highly sensitive to observational errors in its input. This sensitivity rule applies not just to NNs, but to any model as mentioned in the discussion of ill-posed systems in Chapter 9.

The lesson to be learned from this failure is that if the gradient of input with respect to output is small, then output is hypersensitive to input because the gradient of output to input is huge. Thus, careful examination of the Jacobian of the problem can greatly aid in planning a successful regularization strategy. This task can be undertaken either numerically, or by examination of the plotted function as was done above.

Full regularization requires elimination of these zones of hypersensitivity as well as resolution of any regions where the function to be emulated is actually multi-valued. Users of the analytic CMOD4 model do this by limiting analysis to wind speeds of less than some threshold, typically $25\,\mathrm{ms}^{-1}$ (N. S. Winstead 2006, personal communication). It is simple enough to apply this same approach to neural net development, eliminating all cases with higher wind speeds from the training. Note that there are many other forms of regularization. This is just the one that's appropriate for the SAR inverse problem. Note also that, as pointed out in Chapter 9, the resulting NN is apt to be wildly inaccurate outside of its training domain. Thus, in making operational use of the NN, all output wind speeds above $25\,\mathrm{ms}^{-1}$ should be truncated to that value and the value itself viewed as an out-of-domain error flag. This is the current practice with the analytic CMOD4 model, so the NN results shown in Table 10.4 are a fair test of performance relative to the analytic code.

This time the regularization achieves its desired effect. The skill of the fully regularized inverse model is almost as good as that of the forward model. This result holds up when the network is reevaluation on an independent set of test data, demonstrating that over-fitting did not take place. Some hint of the challenge remains in that it took an order of magnitude more training epochs to achieve near perfection on the percent variance explained statistic. The best network yielded an MAE of about $0.1\,\mathrm{ms}^{-1}$, about one tenth of the typical error of the CMOD4 analytic model in comparison with observations (Monaldo et al. 2001).

Table 10.3 Neural net design and performance for the single-valued inverse problem. Those values of r^2 and MAE in parentheses were computed using an independent set of test data, the rest were all computed using a single set of validation data

Network design	Training epochs	$\mathbf{r}^2$	MAE (ms^{-1})	Relative cost
12,6,3	500	0.8521	8.98	0.0913
16,8,4	500	0.8545	7.40	0.1281
24,16,2	500	0.8321	5.05	0.2195
32,16,2	500	0.8697	7.75	0.2713
32,16,4	500	0.8722	7.32	0.2832
32,16,4	5,000	0.8668 (0.8673)	5.37 (5.33)	0.2832

Table 10.4 Neural net design and performance for the fully regularized inverse problem. Those values of r^2 and MAE in parentheses were computed using an independent set of test data, the rest were all computed using a single set of validation data

Network design	Training epochs	r^2	MAE (ms^{-1})	Relative cost
4,2	5,000	0.8416	2.67	0.0235
12,6,3	5,000	0.9857	0.77	0.0913
16,8,4	5,000	0.9878	0.57	0.1281
20,10,5	5,000	0.9894	0.56	0.1681
20,20,2	5,000	0.9984	0.42	0.2208
24,16,2	5,000	0.9988	0.34	0.2195
27,9,3	5,000	0.9974	0.24	0.1886
32,16,2	5,000	0.9982	0.19	0.2713
32,16,2	50,000	0.9990 (0.9990)	0.10 (0.12)	0.2713

Thus, emulation of the CMOD4 analytic model by an appropriate NN would not substantially increase error in operational applications. The remaining question is then, would such a replacement be more efficient in terms of computational cost.

10.3.4 Computational Cost

Estimation of wind speed from SAR backscatter using the analytic CMOD4 forward model is a costly process, both because the model must be run for each candidate wind speed and because the model itself includes a number of calls to costly intrinsic functions such as cosine and hyperbolic tangent. The first problem can be solved by using a NN to emulate the inverse model as discussed above. Unfortunately, from the computational cost perspective, each NN node includes a call to the hyperbolic tangent intrinsic function and so is, itself, fairly costly. An estimate of the computational cost of the existing CMOD4 code and the NN inverse model can be obtained by counting the number of these operations in each approach.

The computational cost for each of the floating point operations and intrinsic function calls is obtained by a benchmarking program that executes each instruction enough times to obtain accurate timing information. The values shown in Table 10.5 were obtained using the default floating point variables in MATLAB version 7 running on a 3.2 GHz Intel Pentium 4 Windows XP PC. These timing results will of course vary with computer language, hardware, and floating point precision, but the final results reported in Tables 10.1 through 10.4 are, to a certain extent, normalized when the NN time estimates are scaled by that for the analytic CMOD4 model. The full operation count for CMOD4 and a multi-layer NN are also shown in Table 10.5.

The key result of this timing analysis is that intrinsic function calls such as cosine and hyperbolic tangent are much more costly than the basic floating point operations of add, multiply, and divide. Thus, the NN times reported in Tables 10.1 through 10.4 are dominated by the hyperbolic tangent calls and are therefore approximately linear with the number of nodes.

Table 10.5 Inputs for the computational cost calculation for CMOD4 and the neural networks. The functions in this table are those used in the analytic CMOD4 code and the neural network

Operation or intrinsic function call	Time in nano-seconds	Uses in CMOD4	Uses in neural network
Add	7	31	Sum over all layers of number of inputs plus one times number of nodes
Multiply	11	36	Sum over all layers of number of inputs times number of nodes
Divide	13	4	0
Cosine	117	2	0
Tangent	174	2	0
Power	477	4	0
Log to base 10	462	1	0
Square root	144	1	0
Hyperbolic tangent	359	2	Equals number of hidden nodes

A 32,16,4 network has 52 such calls and would thus require roughly twice the computer time as a 16,8,2 network. In comparison, the CMOD4 analytic model has only two hyperbolic tangent calls, but does invoke a number of other expensive intrinsic functions.

The key advantage of the neural net inverse model over the analytic CMOD4 model is that it need be run only once instead of 25 times. Thus, even the most complex networks shown in Tables 10.1 through 10.4 require only one fourth of the time of the analytic model. As a result, replacement of the analytic CMOD4 model by a NN inverse model can yield a substantial savings in computational time at the price of a modest increase in wind speed retrieval error.

In light of NN's success on the problem, it is worth considering whether more traditional methods could have achieved a similar speedup in the SAR wind speed retrieval. For example, the search strategy used in the operational CMOD4-based retrieval is to run the analytic forward model at $1\,\mathrm{ms}^{-1}$ intervals from 1 to $25\,\mathrm{ms}^{-1}$. If the more efficient binary search algorithm (Knuth 1997) was used instead, only seven runs of the forward model would be required to bracket the answer to this precision, not 25. This speed ratio of 0.28 is almost identical to the 0.27 achieved by the high accuracy neural net emulation. Thus, there are multiple means of accelerating the SAR wind speed retrieval process. The sole obvious advantage of a NN over binary search is that it does not require branching code and so should pipeline better on modern microprocessors.

10.4 Conclusions

As demonstrated in Chapter 9 and the example above, NNs can provide accurate emulations of smooth nonlinear functions, as long as they are well-posed. NNs can exhibit a substantial speed advantage when computing the desired function would be otherwise cumbersome, for example when the traditional approach involves iterative improvement or examination of many trial solutions. The latter situation occurs for the example problem described above, so a NN with one fourth of the computational cost can emulate the operational retrieval algorithm to within one tenth of the operational model's own accuracy. So, for a slight increase in error and total loss of physical insight, the NN provides a massive improvement in computational efficiency. In contrast, neural net emulation of the forward problem exhibits a substantial cost penalty. This difference between inverse and forward problems results from the efficiency of the analytic forward model, CMOD4, and the inefficiency of the multi-run brute force search strategy traditionally used in applying it to the inverse problem. Interestingly, replacing the brute force search with a binary search using the analytic forward model would yield a run time improvement almost identical to that achieved via NN emulation. Clearly the paths to wisdom are many and varied.

Acknowledgements This work was supported by NSF and ONR through grants ATM-0240869 and N00014-04-10539. Special thanks are due to Nathaniel Winstead and Frank Monaldo of the Johns Hopkins University Applied Physics Laboratory for providing the CMOD4 analytic code and numerous valuable insights.

References

Alpsan, D., Towsey, M., Ozdamar, O., Tsoi, A., & Ghista, D. (1995). Are modified back-propagation algorithms worth the effort? *IEEE International Conference on Neural Networks*, *Orlando, USA*, *1*, 567–571.

Cornford, D., Nabney, I. T., & Ramage, G. (2001). Improved neural network scatterometer forward models. *Journal of Geophysical Research 106*, 22331–22338.

Dodd, N. (1990). Optimization of network structure using genetic techniques. *Proceedings of the International Joint Conference on Neural Networks, Washington D.C., USA*, *1*, 965–970.

Haupt, R., & Haupt, S. (2004). *Practical genetic algorithms* (2nd ed., 253 pp.). Hoboken, NJ: Wiley.

Jones, A. (1993). Genetic algorithms and their applications to the design of neural networks. *Neural Computing and Applications*, *1*, 32–45.

Knuth, D. (1997). *The art of computer programming, volume 3: Sorting and searching* (3rd ed., 780 pp.). Reading, MA: Addison-Wesley.

Krasnopolsky, V., Breaker, L., & Gemmill, W. H. (1995). A neural network as a nonlinear transfer function model for retrieving surface wind speeds from the special sensor microwave imager. *Journal of Geophysical Research 100*, 11033–11045.

Monaldo, F., Thompson, D., Beal, R., Pichel, W., & Clemente-Colón, P. (2001). Comparison of SAR-derived wind speed with model predictions and ocean buoy measurements. *IEEE Transactions on Geoscience and Remote Sensing*, *39*, 2587–2600.

Munro, P. (1993). Genetic search for optimal representations in neural networks. In R. Albrecht, C. Reeves, & N. Steele

(Eds.), *Artificial neural nets and genetic algorithms*. *Proceedings of the international conference* (pp. 628–634). Innsbruck, Austria: Springer.

Nelder, J., & Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, *7*, 308–313.

Reed, R., & Marks, R. (1999). *Neural smithing: Supervised learning in feedforward artificial neural networks* (346 pp.). Cambridge, MA: MIT Press.

Stoffelen, A., & Anderson, D. (1997a). Scatterometer data interpretation: Measurement space and inversion. *Journal of Atmospheric and Oceanic Technology*, *14*, 1298–1313.

Stoffelen, A., & Anderson, D. (1997b). Scatterometer data interpretation: Estimation and validation of the transfer function CMOD4. *Journal of Geophysical Research*, *102*, 5767–5780.

Stogryn, A. P., Butler, C. T., & Bartolac, T. J. (1994). Ocean surface wind retrievals from special sensor microwave imager data with neural networks. *Journal of Geophysical Research*, *90*, 981–984.

Thiria, S., Mejia, C., Badran, F., & Crepon, M. (1993). A neural network approach for modeling nonlinear transfer functions: Application for wind retrieval from spaceborne scatterometer data. *Journal of Geophysical Research*, *98*, 22827–22841.

Witten, I., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (2nd ed., 525 pp.). San Francisco: Morgan Kaufmann.

Yamada, T., & Yabuta, T. (1993). Remarks on neural network controller which uses genetic algorithm. *Proceedings of International Joint Conference on Neural Networks* (pp. 2783–2786). Japan: Nagoya.

Neural Network Applications to Developing Hybrid Atmospheric and Oceanic Numerical Models

11

Vladimir M. Krasnopolsky

List of Acronyms

4-DVar – Four Dimensional Variational (about DAS)
CAM – Community Atmospheric Model
DAS – Data Assimilation System
DIA – Discrete Interaction Approximation
ECMWF – European Center for Medium-Range Weather Forecast
EOF – Empirical Orthogonal Functions
ENM – Environmental Numerical Model
GCM – General Circulation (or Global Climate) Model
HEM – Hybrid Environmental Model
HGCM – Hybrid GCM
HP – Hybrid Parameterization
LWR – Long Wave Radiation LWR
NCAR – National Center for Atmospheric Research
NNIA – Neural Network Interaction Approximation
NNIAE – Neural Network Interaction Approximation using EOF
NSIPP – Natural Seasonal-to-Interannual Predictability Program
Prmse(i) – RMSE for the *i*th profile, see equation (11.10)
PRMSE – Profile RMSE, see equation (11.11)
RMSE – Root Mean Square Error, see equation (11.8)
SWR – Short Wave Radiation

Vladimir M. Krasnopolsky (✉)
Science Application International Company at Environmental Modeling Center, National Centers for Environmental Prediction, National Oceanic and Atmospheric Administration, Camp Springs, Maryland, USA

Earth System Science Interdisciplinary Center, University of Maryland, 5,200 Auth Rd., Camp Springs, MD 20746, USA
Phone: 301-763-8000 ext. 7262; fax: 301-763-8545;
email: vladimir.krasnopolsky@noaa.gov

11.1 Introduction

The past several decades revealed a well pronounced trend in geosciences. This trend marks a transition from investigating simpler linear or weakly nonlinear single-disciplinary systems like simplified atmospheric or oceanic systems that include a limited description of the physical processes, to studying complex nonlinear multidisciplinary systems like coupled atmospheric-oceanic climate systems that take into account atmospheric physics, chemistry, land-surface interactions, etc. The most important property of a complex interdisciplinary system is that it consists of subsystems that, by themselves, are complex systems. Accordingly, the scientific and practical significance of interdisciplinary complex geophysical/environmental numerical models has increased tremendously during the last few decades, due to improvements in their quality via developments in numerical modeling and computing capabilities.

Traditional complex environmental numerical models (ENM) are deterministic models based on "first principle" equations. For example, general circulation models (GCM) a.k.a. global climate models are numerical atmospheric and oceanic models for climate simulation and weather prediction that are based on solving time-dependent 3-D geophysical fluid dynamics equations on a sphere. The governing equations of these models can be written symbolically as,

$$\frac{\partial \psi}{\partial t} + D(\psi, x) = P(\psi, x) \qquad (11.1)$$

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

where ψ are 3-D prognostic or dependent variable or set of variables (e.g., temperature, wind, pressure, moisture); x is a 3-D independent variable (e.g., latitude, longitude, and pressure or height); D is the model dynamics (the set of 3-D partial differential equations of motion, thermodynamics, etc., approximated with a spectral or grid-point numerical scheme); and P is the model physics and chemistry (e.g., the long- and short-wave atmospheric radiation, turbulence, convection and large scale precipitation processes, clouds, interactions with land and ocean processes, etc., and the constituency transport, chemical reactions, etc., respectively). These environmental models are either fully coupled atmosphere-ocean-land/biosphere-chemistry models or partially coupled models (e.g., with the chemistry component calculated off-line, driven by the flow simulated by an atmosphere-ocean-land model).

Another example of a complex ENM is an ocean wind wave model developed for simulation and forecast purposes (Tolman 2002). It is based on a form of the spectral energy or action balance equation

$$\frac{DF}{Dt} = S_{in} + S_{nl} + S_{ds} + S_{sw} \tag{11.2}$$

where F is the 2-D Fourier spectrum of the ocean surface waves, S_{in} is the input source term, S_{nl} is the nonlinear wave-wave interaction source term, S_{ds} is the dissipation or "whitecapping" source term, and S_{sw} represents additional shallow water source terms.

It is important to emphasize that the subsystems of a complex climate (or weather, or ocean) system, such as physical, chemical, and other processes, are so complicated that it is currently possible to include them into GCMs only as 1-D (in the vertical direction) simplified or parameterized versions (a.k.a. parameterizations). These parameterizations constitute the right hand side forcing for the dynamics equations (11.1, 11.2). Some of these parameterizations are still the most time consuming components of ENMs (see examples in the next subsection).

Thus the parameterizations have a very complicated internal structure, are formulated using relevant first principles and observational data, and are usually based on solving deterministic equations (like radiation equations) and some secondary empirical components based on traditional statistical techniques like regression. Accordingly, for widely used state-of-the-art GCMs all major model components (subsystems) are predominantly deterministic; namely, not only model dynamics but the model physics and chemistry are also based on solving deterministic first principle physical or chemical equations.

In the next section, we discuss the concepts of hybrid parameterization (HP) and hybrid environmental models (HEM). HEMs are based on a synergetic combination of deterministic numerical modeling (first principle equations) with NN emulations of some model physics components. We discuss the conceptual and practical possibilities of developing a hybrid GCM (HGCM) and HEM; namely, the possibility of combining accurate and fast NN emulations of model physics components with the deterministic model dynamics of a GCM or ENM, which are the types of complex environmental models used for modern atmospheric and ocean climate modeling and weather prediction.

11.2 Concepts of a Hybrid Model Component and a Hybrid Model

One of the main problems in the development and implementation of modern high-quality high-resolution environmental models is the complexity of the physical, chemical, and other processes involved. Here we will discuss NN emulations for model physics, keeping in mind that the approach is applicable to other model components (chemical, hydrological and other processes) as well. Parameterizations of model physics are approximate schemes, adjusted to model resolution and computer resources, and based on simplified physical process equations and empirical data and relationships. The parameterizations are still so time-consuming, even for the most powerful modern supercomputers, that some of the parameterizations have to be calculated less frequently than the model dynamics. Also, different physical parameterizations are calculated at different frequencies inversely proportional to their computational complexity. This may negatively affect the accuracy of climate and other environmental simulations and predictions.

For example, in the case of a complex GCM, calculation of a physics package (including the atmospheric and land physics) at typical (a few degrees) resolution as in the National Center for Atmospheric Research (NCAR) Community Atmospheric Model (CAM)

takes about 70% of the total model computations. This is despite the fact that while the model dynamics is calculated every 20 min, some computationally expensive parts of the model physics (e.g., short wave radiation) are calculated every hour. The most time consuming calculations of the model atmospheric physics, full long wave radiation including calculation of optical properties, are done only once every 12 h while the heating rates and radiative fluxes are calculated every hour. More frequent model physics calculations, desirable for temporal consistency with model dynamics, and the future introduction of more sophisticated model physics parameterizations will result in a further increase in the computational time spent calculating model physics.

In the wind wave model (11.2), the calculation of the source term, S_{nl} requires roughly 10^3 to 10^4 times more computational effort than all other aspects of the wave model combined. Present operational constraints require that the computational effort for the estimation of S_{nl} should be of the same order of magnitude as for the remainder of the wave model.

This situation is a generic and important motivation in looking for alternative, faster, and most importantly very accurate ways of calculating model physics, chemistry, hydrology and other processes. During the last decade, a new statistical learning approach based on NN approximations or emulations was applied for the accurate and fast calculation of atmospheric radiative processes (e.g., Krasnopolsky (1997); Chevallier et al. (1998)) and for emulations of model physics parameterizations in ocean and atmospheric numerical models (Krasnopolsky et al. 2000, 2002, 2005a, b). *In these works, the calculation of model physics components has been accelerated by 10 to 10^5 times as compared to the time needed for calculating the corresponding original parameterizations of the model physics.*

Approaches formulated by Chevallier et al. (1998, 2000) and Krasnopolsky et al. (2000, 2002, 2005) represent two different ways of introducing a hybridization of first principle and NN components in the physics parameterizations as well as in complex ENMs. These approaches introduce hybridization at two different system levels, at the level of the subsystem (a single parameterization) and at the level of the entire system (ENM). These two approaches lead to the concepts of a hybrid parameterization (HP) (Chevallier et al. 1998, 2000) and a hybrid environmental model (HEM) or hybrid GCM (HGCM) (Krasnopolsky et al. 2000, 2002, 2005; Krasnopolsky and Fox-Rabinovitz 2006a, b). These two concepts are discussed in the following sections.

11.3 Hybrid Parameterizations of Physics

Chevallier et al. (1998, 2000) considered a component of the complex GCM (the ECMWF global atmospheric model) – the long wave radiation (LWR) parameterization. Putting it in terms of the system levels, this single parameterization is considered to be the system and its constituents, with the blocks calculating fluxes, the blocks calculating cloudiness, etc., as the subsystems. The hybridization of first principle components with NN emulations is introduced on the level of these constituents and inside the system, which in this case is the LWR parameterization. A generic LWR parameterization can be represented as a mapping (see Chapter 9, Section 9.1.2),

$$Y = M(X) \tag{11.3}$$

in this particular case the input vector $X = (S,T,V,C)$, where the vector S represents surface variables, T is a vector (profile) of atmospheric temperatures, C is a profile of cloud variables, and the vector V includes all other variables (humidity profile, different gas mixing ratio profiles, etc.). The output of the LWR parameterization, vector Y, is composed of two vectors Q and f, $Y = (Q, f)$. Here Q is a profile of cooling rates $Q = (C_r^1, C_r^2, \ldots, C_r^L)$, where C_r^j is the cooling rate at the j-th vertical model level, and f is a vector of auxiliary fluxes computed by the LWR parameterization. Because of the presence of the cloud variable C, the mapping (11.3) may have finite discontinuities, that is, it is almost continuous.

The ECMWF LWR parameterization considered by Chevallier et al. (1998, 2000) is based on the Washington and Williamson (1977) approach which allows separate cloud variables C. In this parameterization, level fluxes are calculated as,

$$F(S, T, V, C) = \sum_i \alpha_i(C) F_i(S, T, V) \tag{11.4}$$

where each partial or individual flux $F_i(S, T, V)$ is a continuous mapping and all discontinuities

related to the cloudiness are included in $\alpha_i(C)$. In their hybrid parameterization "NeuroFlux", Chevallier et al. (1998, 2000) combined calculations of cloudiness functions $\alpha_i(C)$ based on first principle equations with NN approximations for a partial or individual flux $F_i(S, T, V)$. Thus, the flux at each level (11.4) is a linear combination of approximating NNs and cloud physics coefficients $\alpha_i(C)$. As the result, the "NeuroFlux" hybrid LWR parameterization developed by Chevallier et al. (1998, 2000) is a battery of about 40 NNs (two NNs – one for the upward and another one for the downward radiation fluxes – for each of vertical level where clouds are possible). To calculate "NeuroFlux" outputs, namely the cooling rates C_rs, linear combinations of the individual approximating NNs F (equation 11.4) are differentiated at each vertical level,

$$C_r(P) = \frac{\partial F(P)}{\partial P}, \tag{11.5}$$

where P is atmospheric pressure.

The "NeuroFlux" has a very good accuracy; its bias is about 0.05 K/day and RMS error is about 0.1 K/day compared to the LWR parameterization by Washington and Williamson (1977). It is eight times faster than the parameterization by Washington and Williamson (1977). This HP approach has already led to the successful operational implementation of "NeuroFlux" in the ECMWF 4-DVar data assimilation system.

As for limitations of the HP approach, the main one stems from a basic feature of the HP approach; it is based on the analysis of the internal structure of a particular parameterization. The final design of HP is based on and follows this internal structure. Because all parameterizations have different internal structures, the approach and design of a HP developed for one parameterization usually cannot be used, without significant modifications, for another parameterization. For example, the approach used by Chevallier et al. (1998, 2000) and the design of the HP "NeuroFlux" is completely based on the possibility of separating the dependence on the cloudiness (see equation 11.4). Many other LWR parameterizations, like the NCAR CAM LWR parameterization (Collins 2001; Collins et al. 2002) or the LWR parameterization developed by Chou et al. (2001), do not allow for such separation of variables. Thus, for these LWR parameterizations as well as the short wave radiation (SWR) and the moisture model physics block parameterizations, the HP approach developed by Chevallier et al. (1998, 2000) cannot be applied directly; it should be significantly modified or redesigned for each particular new parameterization.

11.4 Hybrid Numerical Models: Accurate and Fast NN Emulations for Parameterizations of Model Physics

A new concept of a complex hybrid environmental model (HEM) has been formulated and developed by Krasnopolsky et al. (2000, 2002, 2005a) and by Krasnopolsky and Fox-Rabinovitz (2006a, b). The hybrid modeling approach considers the whole GCM or ENM as a system. Dynamics and parameterizations of physics, chemistry, etc., are considered to be the components of the system. Hybridization in this case is introduced at the level of components inside the system (ENM). For example, the entire LWR (or SWR) parameterization is emulated by a single NN as a single/elementary object or block. The NN emulation approach is based on the general fact that any parameterization of model physics can be considered as a continuous or almost continuous mapping (11.3) (see Chapter 9, Section 3.1.2).

Here we use the NCAR CAM (see *Journal of Climate* (1998) for the description of the model), a widely recognized state-of-the-art GCM used by a large modeling community for climate predictions, and the state-of-the-art NCEP wind wave model (Tolman 2002) as examples of a complex GCM and ENM. After applying the hybridization approach to the first principle based components of these models by developing NN *emulations* of model physics parameterizations, these models become the examples of an HGCM and HEM, correspondingly.

Krasnopolsky and Fox-Rabinovitz (2006a, b) formulated a developmental framework and test criteria that can be recommended for developing and testing the statistical learning components of HGCM, i.e., NN emulations of model physics components. The developmental process consists of three major steps:

1. Problem analysis or analysis of the model component (i.e., the original parameterization) to be approximated to determine the optimal structure and configuration of the NN emulations – the number of inputs and outputs and the first guess of the functional complexity of the original parameterization that determines an initial number of hidden neurons in one hidden layer of (see Chapter 9, Eqs. 9.2, 9.3).
2. Generation of representative data sets for training, validation, and testing. This is achieved by using data for NN training that are simulated by running an original GCM, i.e., a GCM with the original parameterization. When creating a representative data set, the original GCM must be run long enough to produce all possible atmospheric model simulated states, phenomena, etc. Here, due to the use of simulated data, it is not a problem to generate the sufficiently representative (and even redundant) data sets required to create high quality NN emulations. Using model-simulated data for NN training allows a high accuracy of emulation to be achieved because simulated data are almost free of the problems typical in empirical data (like a high level of observational noise, sparse spatial and temporal coverage, poor representation of extreme events, etc.).
3. Training the NN. Several different versions of NNs with different architectures, initialization, and training algorithms should be trained and validated. As for the NN architecture, the number of hidden neurons k should be kept to the minimum number that provides a sufficient emulation accuracy to create the high quality NN emulations required.

Testing the HGCM that uses the trained NN emulation consists of two major steps. The *first step* is testing the accuracy of the NN approximation against the original parameterization using the independent test data set. In the context of the hybrid approach, the accuracy and improved computational performance of NN emulations, and eventually the HGCM is always measured against the corresponding controls, namely the original parameterization and its original GCM. Both the original parameterization and its NN emulation are complicated multidimensional mappings. Many different statistical metrics of the emulation accuracy should be calculated to assure that a sufficiently complete evaluation of the emulation accuracy is obtained. For example, total, level, and profile statistics have to be evaluated (see Section 11.5.1). The *second* test *step* consists of a comprehensive comparison and analysis of parallel HGCM and GCM runs. For the parallel model simulations all relevant model prognostic (i.e., time-dependent model variables) and diagnostic fields should be analyzed and carefully compared to assure that the integrity of the original GCM and its parameterization, with all its details and characteristic features, is precisely preserved when using a HGCM with NN emulation (see Section 11.5). This test step involving model simulations is crucially important. GCMs are essentially nonlinear complex systems; in such systems, small systematic, and even random, approximation errors can accumulate over time and produce a significant impact on the quality of the model results. Therefore, the development and application framework of the new hybrid approach should be focused on obtaining a high accuracy in both NN emulations and HGCM simulations.

11.5 Atmospheric Applications: NN Emulation Components and HGCM

The NCAR CAM and NASA NSIPP (Natural Seasonal-to-Interannual Predictability Program) GCM are used in this section as examples of GCMs. The NCAR CAM is a spectral model that has 42 spectral components (or approximately $3^\circ \times 3.5^\circ$ horizontal resolution) and 26 vertical levels. The NSIPP model is a grid point GCM that has $2^\circ \times 2.5^\circ$ latitude $\times$ longitude horizontal resolution and 40 vertical levels. Note that the model vertical levels are distributed between the surface and upper stratosphere, which is at approximately 60–80 km. NN emulations were developed for the two most time consuming components of model physics, LWR and short wave radiation (SWR). The NCAR and NSIPP models have different LWR and SWR parameterizations. The complete description of the NCAR CAM atmospheric LWR is presented by Collins (2001) and Collins et al. (2002), and the NSIPP LWR by Chou et al. (2001). The full model radiation (or total LWR and SWR) calculations take $\sim$70% of the total model physics calculations. It is noteworthy that the results presented in this section were obtained using the two latest versions of NCAR CAM – the CAM-2 and CAM-3. The version of CAM used in the

calculations is specified in the corresponding subsections below.

11.5.1 NCAR CAM Long Wave Radiation

The function of the LWR parameterization in atmospheric GCMs is to calculate the heating fluxes and rates produced by LWR processes. As was already mentioned, the entire LWR parameterization can be represented as an almost continuous mapping (equation 11.3). Here a very general and schematic outline of the internal structure of this parameterization is given in order to illustrate the complexity that makes it a computational "bottleneck" in the NCAR CAM physics. This information about the internal structure of the LWR parameterization was not used when creating the LWR NN emulation.

The method for calculating LWR in the NCAR CAM is based on LW radiative transfer equations in an absorptivity/emissivity formulation (see Collins 2001 and references there),

$$F^{\downarrow}(p) = B(p_t) \cdot \varepsilon(p_t, p) + \int_{p_t}^{p} \alpha(p, p') \cdot dB(p')$$

$$F^{\uparrow}(p) = B(p_s) - \int_{p}^{p_s} \alpha(p, p') \cdot dB(p') \qquad (11.6)$$

where $F^{\uparrow}(p)$ and $F^{\downarrow}(p)$ are the upward and the downward heat fluxes, $B(p) = \sigma \cdot T^4(p)$ is the Stefan-Boltzmann relation; pressures p_s and p_t refer to the top and surface atmospheric pressures, and α and ε are the atmospheric absorptivity and emissivity. To solve the integral equations (11.6), the absorptivity and emissivity have to be calculated by solving the following integro-differential equations,

$$a(p, p') = \frac{\int_0^{\infty} \{dB_{\nu}(p')/dT(p')\} \cdot [1 - \tau_{\nu}(p, p')] \cdot d\nu}{dB(p)/dT(p)}$$

$$\varepsilon(p_t, p) = \frac{\int_0^{\infty} B_{\nu}(p_t) \cdot [1 - \tau_{\nu}(p_t, p)] \cdot d\nu}{B(p_t)} \qquad (11.7)$$

where the integration is over wave number ν, and $B \cdot (p_t)$ is the Planck function. To solve equations (11.7) for the absorptivity and emissivity, additional calculations have to be performed and the atmospheric transmission τ_{ν} has to be calculated. This calculation involves a time consuming integration over the entire spectral range of gas absorption. Equations (11.6, 11.7) illustrate the complexity of the LWR internal structure and explain the poor computational performance of the original NCAR CAM LWR parameterization, which in this case is determined by the mathematical complexity of the original LWR parameterization.

The input vectors for the NCAR CAM LWR parameterization include ten vertical profiles (atmospheric temperature, humidity, ozone, CO_2, N_2O, CH_4, two CFC mixing ratios (the annual mean atmospheric mole fractions for halocarbons), pressure, and cloudiness) and one relevant surface characteristic (upward LWR flux at the surface). The CAM LWR parameterization output vectors consist of the vertical profile of heating rates (HRs) and several radiation fluxes, including the outgoing LWR flux from the top layer of the model atmosphere (the outgoing LWR or OLR). The NN emulation of the NCAR CAM LWR parameterization has the same number of inputs (220 total) and outputs (33 total) as the original NCAR CAM LWR parameterization.

NCAR CAM was run for 2 years to generate representative data sets. The first year of the model simulation was divided into two independent parts, each containing input/output vector combinations. The first part was used for training and the second for validation (control of overfitting, control of a NN architecture, etc.). The second year of the simulation was used to create a test data set completely independent from both the training and validation sets. This data set was used for testing only. All approximation statistics presented in this section were calculated using this independent test data set.

The NN emulations developed were tested against the original NCAR CAM LWR parameterization. Both the original LWR parameterization and its NN emulation are complex multidimensional mappings. Because of their complexity, many different statistics and statistical cross-sections were calculated to obtain a complete enough comparison between these two objects and to evaluate the accuracies of the NN emulations. The mean difference B (bias or systematic error of approximation) and the root mean square difference *RMSE* (a root mean square error of approximation)

between the original parameterization and its NN emulation are calculated as follows:

$$B = \frac{1}{N \times L} \sum_{i=1}^{N} \sum_{j=1}^{L} [Y(i, j) - Y_{NN}(i, j)]$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} \sum_{j=1}^{L} [Y(i, j) - Y_{NN}(i, j)]^2}{N \times L}} \tag{11.8}$$

where $Y(i, j)$ and $Y_{NN}(i, j)$ are outputs from the original parameterization and its NN emulation, respectively, where i = *(latitude, longitude)*, $i = 1, \ldots, N$ is the horizontal location of a vertical profile; N is the number of horizontal grid points; and $j = 1, \ldots, L$ is the vertical index where L is the number of the vertical levels.

These two error characteristics (equations (11.8)) describe the accuracy of the NN emulation integrated over the entire 4-D (latitude, longitude, height, and time) data set. Using a minor modification of equations (11.8), the bias and RMSE for the mth vertical level of the model can be calculated:

$$B_m = \frac{1}{N} \sum_{i=1}^{N} [Y(i, m) - Y_{NN}(i, m)]$$

$$RMSE_m = \sqrt{\frac{\sum_{i=1}^{N} [Y(i, m) - Y_{NN}(i, m)]^2}{N}} \tag{11.9}$$

The root mean square error can also be calculated for each ith profile:

$$prmse(i) = \sqrt{\frac{1}{L} \sum_{j=1}^{L} [Y(i, j) - Y_{NN}(i, j)]^2} \tag{11.10}$$

This error is a function of the horizontal location of the profile. It can be used to calculate a mean profile root mean square error *PRMSE* and its standard deviation σ_{PRMSE} which are location independent:

$$PRMSE = \frac{1}{N} \sum_{i=1}^{N} prmse(i)$$

$$\sigma_{PRMSE} = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} [prmse(i) - PRMSE]^2} \tag{11.11}$$

The statistics (11.11) and (11.8) both describe the accuracy of the NN emulation integrated over the entire 4-D data set. However, because of a different order of integration it reveals different and complementary information about the accuracy of the NN emulations. The root mean square error profile can be calculated:

$$prmse(j) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} [Y(i, j) - Y_{NN}(i, j)]^2} \tag{11.12}$$

Several NNs have been developed that all have one hidden layer with 20 to 300 neurons. Varying the number of hidden neurons allows one to demonstrate the dependence of the accuracy of NN emulation on this parameter, which is actually the complexity of the NN emulation, as well as selecting an optimal NN emulation (Krasnopolsky et al. 2005) with the minimal complexity that still provides an emulation accuracy sufficient for a successful multi-decadal climate model integration.

All NN emulations (Krasnopolsky et al. 2005; Krasnopolsky and Fox-Rabinovitz 2006a, b) developed for the NCAR CAM LWR have almost zero or negligible systematic errors (biases). Figure 11.1 illustrates convergences of root mean square errors (11.8, 11.9, and 11.11) that are random errors in the case of negligible biases. The figure shows that an error convergence has been reached when the number of hidden neurons $k \approx 100$. However, the convergence becomes slow and non-monotonic at $k \approx 50$. The final decision about the optimal NN emulation (in terms of sufficient accuracy and minimal complexity) to be implemented into the model is based on decadal (40 year) integrations using the NN emulations within HGCM (Krasnopolsky et al. 2005; Krasnopolsky and Fox-Rabinovitz 2006a, b). For assessing the impact of using an NN emulation of the LWR parameterization in the HGCM, parallel climate simulation runs were performed with the original GCM (NCAR CAM including the original LWR parameterization) as the control run and with the HGCM (NCAR CAM including the NN emulations of LWR described above). The climate simulations were run for 50 years. As is usually done in climate simulations the simulated fields for the first 10 years, that potentially include the climate model spin-up effects, are not used for the

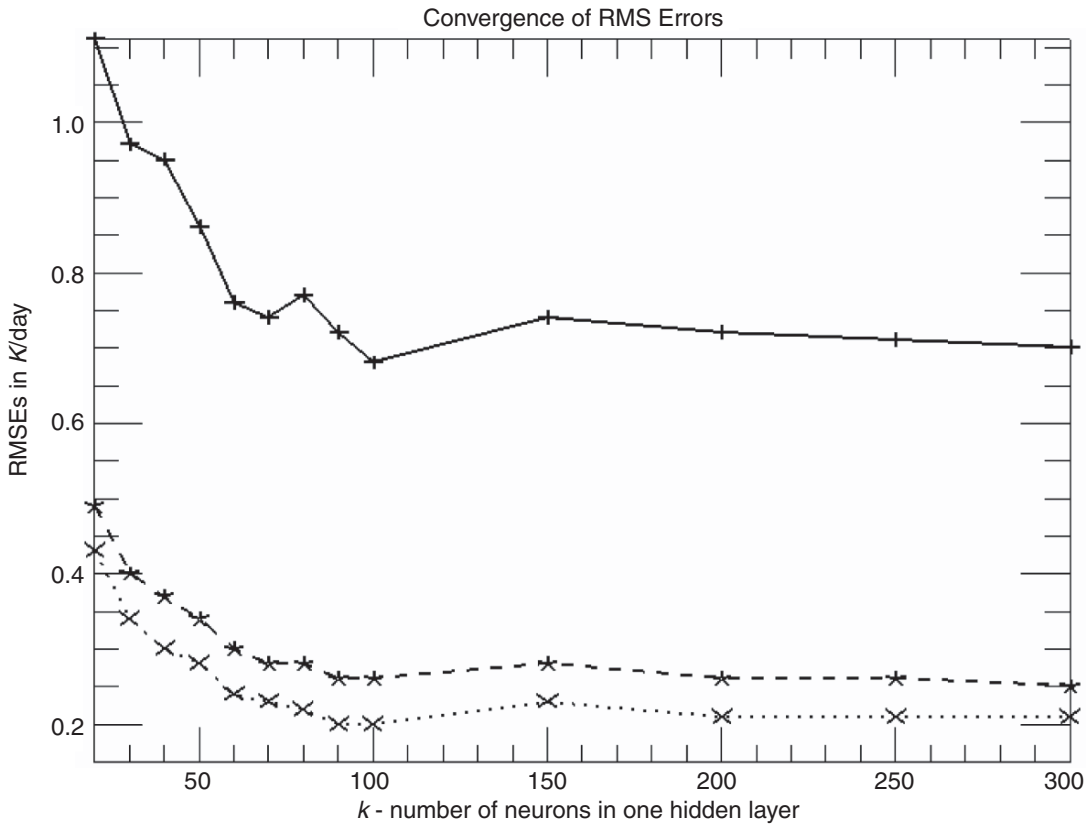


Fig. 11.1 The convergence of root mean square errors (11.8, 11.9, and 11.11). Solid line – $RMSE_{26}$ (11.9) dashed line – RMSE (11.8), and dotted line – PRMSE (11.11)

analysis of the simulation results, leaving the remaining 40 year period to be used for that purpose.

The NN emulation with $k = 50$ (NN50) is the simplest NN emulation that could be integrated into the model for decadal (40 years or longer) climate simulations without any visible (significant) accumulations of errors in climate simulations, compared to the control run with the original LWR parameterization. This is the main indicator (in the framework of this NN application) that the accuracy of this NN emulation is sufficient for this application. Figure 11.2 shows the vertical error profile (11.12) *prmse(j)* for the "optimal" NN emulation with 50 hidden neurons (NN50). It shows that the errors are very small; at the top 10 levels the error does not exceed 0.2 K/day, at the top 20 levels it does not exceed 0.3 K/day and reaches just about 0.6–0.8 K/day at the lowest level, which does not lead to significant errors in the 40 year climate simulations with HGCM. In addition to having sufficient emulation accuracy, the NN50 NN emulation performs about 150 times faster than the original NCAR CAM LWR parameterization in a code by code comparison.

Comparisons between the control and NN emulation runs are presented in Table 11.1. They are done by analyzing the time (40-year) and global mean differences between the results of the parallel runs, as is routinely done in climate modeling. In the climate simulations performed with the original GCM and with HGCM, the time and global mean mass or mean surface pressure are precisely preserved, which is the most important preservation property for climate simulations. For the NN50 run, there is no difference in mean sea surface pressure between the NN and control runs (see Table 11.1). Other time global means, some of which are also presented in Table 11.1, show a profound similarity between the parallel simulations for these terms. These very small differences indicate the very close results from the parallel climate simulations. Other

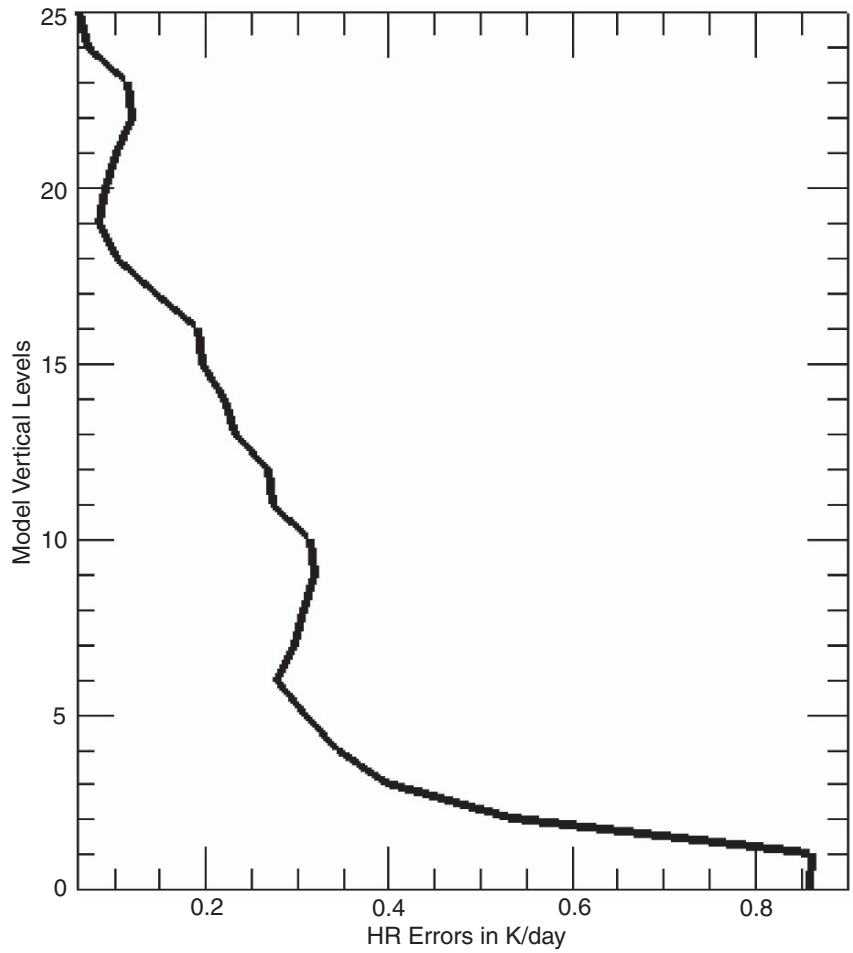


Fig. 11.2 The vertical error profile (11.10), *prmse(j),* for the "optimal" LWR NN emulation with 50 hidden neurons (NN50)

simulations (with NN90, NN150, NN200, etc.) also show that the HGCM results are profoundly similar to those of the original GCM (Krasnopolsky et al. 2005; Krasnopolsky and Fox-Rabinovitz 2006a, b). It is noteworthy that the differences between these parallel runs (HGCM and GCM) do not exceed the differences seen in two identical GCM runs performed on different supercomputers.

11.5.2 NASA NSIPP Long Wave Radiation

The robustness of the NN emulation approach was investigated using another GCM. The NASA NSIPP GCM (with a different LWR parameterization and other different model components compared to the NCAR CAM and its LWR parameterization) was used for this purpose. The input vector for the NSIPP LWR

Table 11.1 Time (40-years) and global means for mass (mean sea level pressure) and other model diagnostics for the NCAR CAM-2 climate simulations with the original LWR parameterization (in GCM), and its NN emulation (in HGCM) using NN50 and their differences (in %)

Field	GCM with the original LWR parameterization	HGCM with NN emulation	Difference (in %)
Mean sea level pressure (hPa)	1,011.48	1,011.48	$<10^{-3}$
Surface temperature (K)	289.02	288.97	0.02
Total precipitation (mm/day)	2.86	2.89	1.04
Total cloudiness (fractions, %)	60.71	61.26	0.9
Wind at 12 km (m/s)	16.21	16.16	0.3

includes surface temperature and five vertical profiles of cloud fraction, pressure, temperature, specific humidity and ozone mixing rate, for a total of 202 inputs. The NSIPP LWR output vector consists of a profile of heating rates and one surface parameter, for a total of 41 outputs.

The NN emulation accuracy and complexity results in this case (Krasnopolsky et al. 2005; Krasnopolsky and Fox-Rabinovitz 2006a, b) are very similar to the ones presented above for NCAR CAM. This illustrates the robustness of the NN emulation approach.

11.5.3 NCAR CAM Short Wave Radiation

The second component of atmospheric radiation is short wave radiation (SWR). LWR and SWR together comprise the total atmospheric radiation. The function of the SWR parameterization in atmospheric GCMs is to calculate the heating fluxes and rates produced by SWR processes. A description of the NCAR CAM atmospheric SWR parameterization is presented in a special issue of *Journal of Climate* (1998). The input vectors for the NCAR CAM SWR parameterization include twenty-one vertical profiles (specific humidity, ozone concentration, pressure, cloudiness, aerosol mass mixing ratios, etc.) and several relevant surface characteristics. NN emulations for the CAM-2 and CAM-3 versions of NCAR CAM SWR parameterizations have been developed (Krasnopolsky and Fox-Rabinovitz 2006a, b). The major difference between the CAM-2 and CAM-3 SWR versions is that CAM-3 uses significantly more information about aerosols. This extended aerosol information is responsible for a substantial increase in the number of inputs into the CAM-3 SWR parameterization as compared with CAM-2. The CAM SWR parameterization output vectors consist of a vertical profile of heating rates (HRs) and several radiation fluxes.

The data sets for training, validating, and testing SWR emulating NNs were generated in the same way as those for the LWR NN emulations described above. SWR NN emulations were tested against the original NCAR CAM SWR parameterizations using the independent test set.

The NN emulations of NCAR CAM-2 and CAM-3 SWR parameterizations have 173 and 451 inputs, respectively, and 33 outputs, which are the same numbers as the inputs and outputs for the original NCAR CAM-2 and CAM-3 SWR parameterizations. As in the case of the LWR parameterizations, several NNs were developed that all have one hidden layer with 20 to 300 neurons. In the case of the SWR parameterizations, the convergence of root mean square errors (11.8, 11.9, and 11.11) is very similar to that for the LWR parameterization shown in Fig. 11.1. The convergence is reached when the number of hidden neurons $k \approx 100$. However, it becomes slow and non-monotonic at $k \approx 50$. The NN emulation with $k = 55$ (NN55) is the simplest NN emulation that satisfies the sufficient accuracy criterion; it could be integrated in the HGCM for multi-decadal simulations without visible (significant) accumulations of errors in climate simulations as compared to the control run with the original SWR parameterization. Figure 11.3 shows the vertical error profile (11.12) *prmse(j)* for the "optimal" NN emulation NN55. It shows that the errors are very small; at the top 20 levels the error does not exceed 0.2 K/day and reaches just about 0.45 K/day at the lowest level, which does not lead to significant errors in the HGCM climate simulations. In addition to sufficient emulation accuracy, the NN55 SWR NN emulation performs about 20 times faster than the original NCAR CAM SWR parameterization in a code by code comparison.

Comparisons between the control and NN emulation runs are also presented in Table 11.2 (see the Table 11.1 explanation in the text above). For the NN55 run there is a negligible difference between the NN and control runs for sea surface pressure (see Table 11.2). Other time global means, some of which are also presented in Table 11.2, show a profound similarity between the parallel simulations for these terms, with differences usually within about 0.3%. These very small differences indicate the very close results from the parallel climate simulations. Other simulations (with NN100, NN150, etc.) also show that the HGCM results are profoundly similar to those of the original GCM (Krasnopolsky and Fox-Rabinovitz 2006a, b).

11.5.4 NCAR CAM Full Radiation

It was shown in the previous subsections that both components of radiation, LWR and SWR, can be successfully emulated using the NN approach. This means

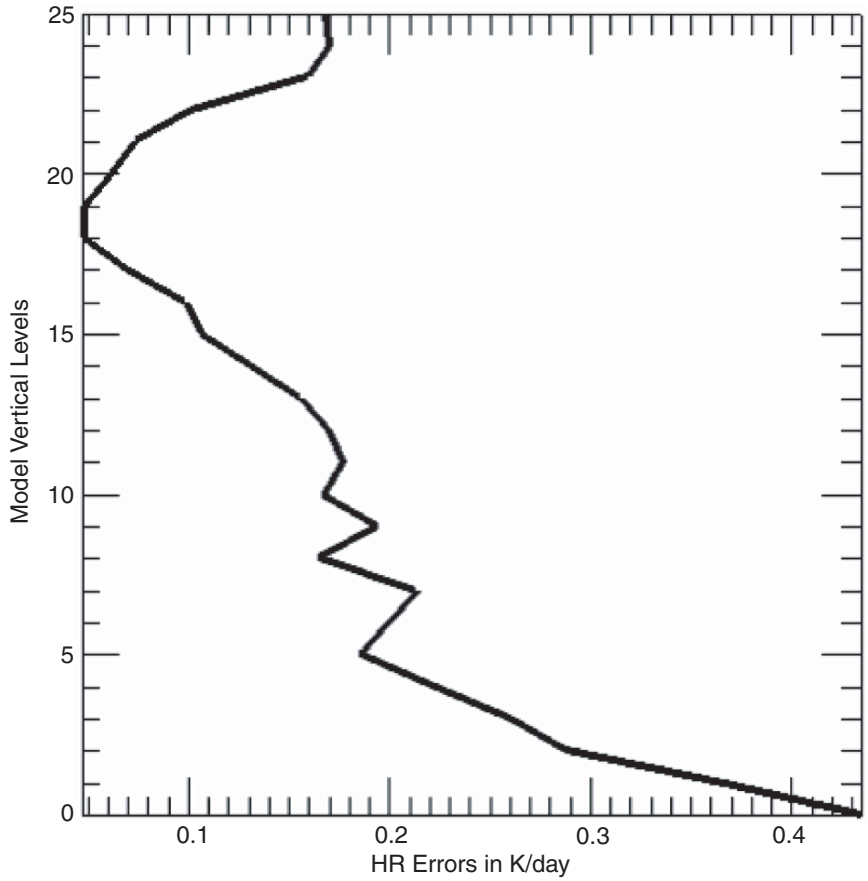


Fig. 11.3 The vertical error profile (11.10), *prmse(j),* for the "optimal" SWR NN emulation with 55 hidden neurons (NN55)

that these most time consuming components of model physics can be significantly sped up without any negative impact on the accuracy of the climate simulations. The next logical step is to combine these two NN emulations (LWR and SWR) to emulate the total model radiation. The NN50 LWR emulation and NN55 SWR emulation described in the previous subsections were combined together in one HGCM. This HGCM with the NN emulations of the total model radiation was integrated for 40 years and the results of the climate simulation were compared with those of the NCAR CAM-2 GCM simulation control run with the original NCAR CAM LWR and SWR parameterizations. In addition to having a sufficient emulation accuracy, the total radiation NN emulations perform about 12 times faster in the model than the original NCAR CAM parameterizations in terms of model time spent to calculate total radiation.

Comparisons between the control and NN emulation runs are presented in Table 11.3 (see the Table 11.1 explanation in the text above). For the total radiation run there is a negligible difference of

Table 11.2 Time (40-years) and global means for model diagnostics from NCAR CAM-2 climate simulations with the original SWR (in GCM), its NN emulation (in HGCM) using NN55, and their differences (in %)

Field	GCM with the original SWR parameterization	HGCM with SWR NN emulation	Difference (in %)
Mean sea level pressure (hPa)	1,011.48	1,011.49	0.001
Surface temperature (K)	289.01	288.97	0.01
Total precipitation (mm/day)	2.86	2.86	<0.1
Total cloudiness (fractions, %)	60.73	60.89	0.3
Wind at 12 km (m/s)	16.21	16.20	0.06

Table 11.3 Time (40-years) and global means for model diagnostics from NCAR CAM-2 climate simulations with the original LWR and SWR (in GCM), their NN emulations (in HGCM) using NN50 (LWR) and NN55 (SWR), and their differences (in %)

Field	GCM with the original LWR and SWR parameterizations	HGCM with LWR and SWR NN emulations	Difference (in %)
Mean sea level pressure (hPa)	1,011.48	1,011.50	0.002
Surface temperature (K)	289.02	288.92	0.03
Total precipitation (mm/day)	2.86	2.89	1.04
Total cloudiness (fractions, %)	60.71	61.12	0.6
Wind at 12 km (m/s)	16.21	16.29	0.5

0.002% between the NNs and control runs for sea surface pressure (see Table 11.3). Other time global means, some of which are also presented in Table 11.3, show a profound similarity between the parallel simulations for these terms. These very small differences indicate the very close results from the parallel climate simulations.

11.6 Ocean Application of the Hybrid Model Approach: Neural Network Emulation of Nonlinear Interactions in Wind Wave Models

The ocean wind wave model used for simulation and forecast purposes is another example of an ENM. It is based on a form of the spectral energy or action balance equation (11.2) and has the nonlinear wave-wave interaction source term S_{nl} as a part of the model physics in the right hand side of the equation. In its full form (e.g., Hasselmann and Hasselmann 1985) the calculation of the S_{nl} interactions requires the integration of a six-dimensional Boltzmann integral:

$$\begin{aligned} S_{nl}(\vec{k}_4) &= T \otimes F(\vec{k}) \\ &= \omega_4 \int G(\vec{k}_1, \vec{k}_2, \vec{k}_3, \vec{k}_4) \cdot \delta(\vec{k}_1 + \vec{k}_2 - \vec{k}_3 - \vec{k}_4) \\ &\times \delta(\omega_1 + \omega_2 - \omega_3 - \omega_4) \left[n_1 \cdot n_3 \cdot (n_4 - n_2) \right. \\ &\left. + n_2 \cdot n_4 \cdot (n_3 - n_1) \right] d\vec{k}_1 d\vec{k}_2 d\vec{k}_1 \qquad (11.13) \\ n(\vec{k}) &= \frac{F(\vec{k})}{\omega}; \quad \omega^2 = g \cdot k \cdot \tanh(kh) \end{aligned}$$

where the complicated coupling coefficient G contains moving singularities; T is a symbolic representation for the mapping. This integration requires roughly 10^3 to 10^4 times more computational effort than all other aspects of the wave model combined. Present operational constraints require that the computational effort for the estimation of S_{nl} should be of the same order of magnitude as the remainder of the wave model. This requirement was met with the development of the Discrete Interaction Approximation (DIA, Hasselmann et al. 1985). Two decades of experience with the DIA in wave models has identified significant shortcomings in the DIA (Tolman et al. 2005). It is not sufficiently accurate and, in many physically important cases, significantly deviates from the equation (11.13) deteriorating the prediction capabilities of the wind wave model (11.2).

When considering the above, it is crucially important for the development of third generation wave models to develop *an economical yet accurate approximation* for S_{nl}. A Neural Network Interaction Approximation (NNIA) was explored to achieve this goal (Krasnopolsky et al. 2002; Tolman et al. 2005). NNs can be applied here because the nonlinear interaction (11.13) is essentially a nonlinear mapping, symbolically represented by T, which relates two vectors F and S_{nl} (2-D fields in this case). Discretization of S and F (as is necessary in any numerical approach) reduces (11.13) to a continuous mapping of two vectors of finite dimensions. Modern high resolution wind wave models use descretization on a two dimensional grid which leads to S and F vector dimensions on the order of $N \sim 1{,}000$. It seems unreasonable to develop a NN emulation of such a high dimensionality (about 1,000 inputs and outputs). Moreover, such a NN will be grid dependent.

In order to reduce the dimensionality of the NN and convert the mapping (11.13) to a continuous mapping of two finite vectors that are less dependent on the actual spectral discretization, the spectrum F and source function S_{nl} are expanded using systems of

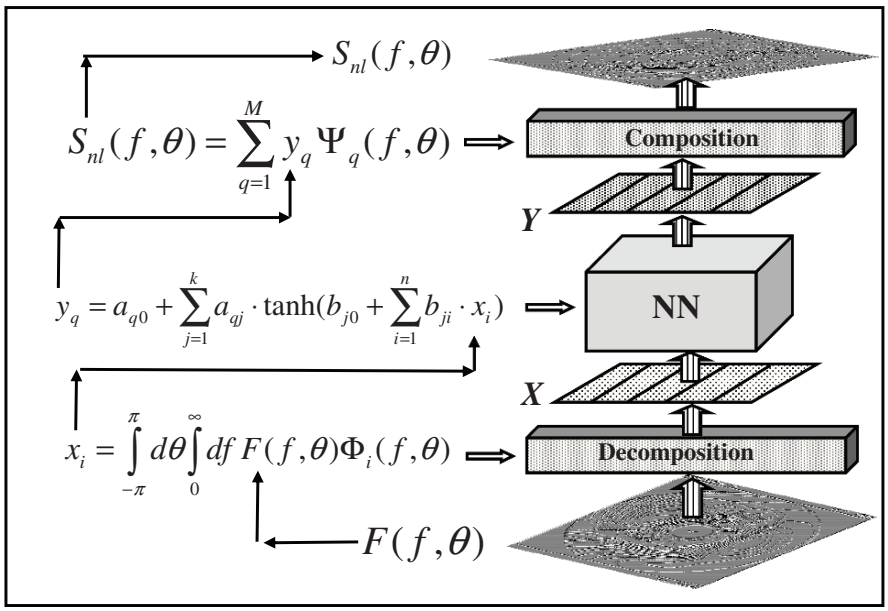


Fig. 11.4 Graphical representation of the NNIA and NNIAE algorithms

two-dimensional functions, each of which (Φ_i and Ψ_q) creates a complete and orthogonal two-dimensional basis

$$F \approx \sum_{i=1}^{n} x_i \Phi_i, \qquad S_{nl} \approx \sum_{q=1}^{m} y_q \Psi_q, \tag{11.14}$$

where for the coefficients of decomposition/composition x_i and y_q,

$$x_i = \iint F \Phi_i, \qquad y_q = \iint S_{nl} \Psi_q, \tag{11.15}$$

where the double integral identifies integration over the spectral space. Now, the developed NN emulation relates vectors of coefficients $\boldsymbol{X}$ and $\boldsymbol{Y}$: $\boldsymbol{Y} = \boldsymbol{T}_{NN}(\boldsymbol{X})$.

To train the NN emulation T_{NN}, a training set has to be created that consists of pairs of the vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$. To create this training set, a representative set of spectra F_p has to be generated with corresponding (exact) interactions $S_{nl,p}$ using equation (11.13). For each pair $(F, S_{nl})_p$, the corresponding vectors $(\boldsymbol{X},\boldsymbol{Y})_p$ are determined using equation (11.15). These pairs of vectors are then used to train the NN to obtain T_{NN}. After T_{NN} has been trained, the resulting NN Interaction Approximation (NNIA) algorithm consists of three steps: (i) decompose the input spectrum F by applying equation (11.15) to calculate $\boldsymbol{X}$; (ii) estimate $\boldsymbol{Y}$ from $\boldsymbol{X}$ using NN; and (iii) compose the output source function S_{nl} from $\boldsymbol{Y}$ using equation (11.14). A graphical representation of the NNIA algorithm is shown in Fig. 11.4.

Two approaches have been used for the basis functions. The first is the mathematical basis used in Krasnopolsky et al. (2002). As is usually done in the parametric spectral description of wind waves, separable basis functions are chosen where the frequency and angular dependence are separate. The advantage of this choice of basis functions is the simplicity of the basis generation. The disadvantage is the slow convergence of the decompositions. As an alternative, a second approach to the basis functions choice has been investigated. In this approach Empirical Orthogonal Functions (EOFs) or principal components (Lorenz 1956; Jolliffe 2002) are used (Tolman et al. 2005).

EOFs compose a statistically optimal basis. In the case considered, the basis functions Φ_i and Ψ_q are functions of two variables f and θ. The set of spectra F and source terms S_{nl}, which are used for the training of the NN, are also used to generate the EOFs for decomposing F and S_{nl}. When using EOFs the basis generation procedure is computationally expensive, with the cost increasing as the resolution of the model increases. However, as in NN training the basis generation needs to be performed only once. Stored results can be used without the need to recalculate in the final NNIA algorithm. The main advantage of EOFs is the fast convergence of the decomposition.

Table 11.4 Approximation RMSes (in nondimensional units) and performance (DIA calculation time is selected as a unit) for DIA, NNIA, NNIAE, and exact S_{nl} calculation (original)

Algorithm	RMSE	Performance
DIA	0.312	1
NNIA	0.088	4
NNIAE	0.035	7
Original parameterization	0.	$\sim 8. \times 10^5$

To distinguish between NN algorithms using different basis functions for decomposition, we use the abbreviation NNIAE for our NN algorithm that used the EOF basis. Table 11.4 demonstrates comparisons of the accuracy and performance of DIA with the two NN emulations NNIA and NNIAE, all versus the exact calculation of S_{nl} original parameterization. Approximation errors (RMSEs) are calculated in nondimensional units and performance is measured in DIA calculation times (taken as a unit). The NNIAE is nearly ten times more accurate than DIA. It is about 10^5 times faster than the original parameterization. As in the case of the atmospheric long wave radiation, a careful investigation of the parallel runs with the original ENM (the wave model with the original wave-wave interaction) and the HEM run with the NN emulation should be performed for the final test of the NN emulation (Tolman et al. 2005).

11.7 Discussion

11.7.1 Summary and Advantages of the Hybrid Modeling Approach

In this chapter, we reviewed a new hybrid paradigm in environmental numerical modeling. Within the framework of this paradigm a new type of ENM – a hybrid environmental model (HEM) based on a synergetic combination of deterministic modeling and statistical learning within an HEM (using a NN technique) is introduced. This approach uses NNs to develop highly accurate and fast emulations of model physics components. The presented results show:

(i) The conceptual and practical possibility of developing HEMs with accurate NN emulations of model components, which preserve the integrity and all the detailed features of the original ENM.

(ii) NN emulations of model physics parameterizations developed by Krasnopolsky et al. (2000, 2002, 2005) are practically identical to the original physical parameterizations, due to the capability of NN techniques to very accurately emulate complex systems like the model physics. This fact allows the integrity and level of complexity of the state-of-the-art parameterizations of model physics to be preserved. As a result, for example, a HGCM using these NN emulations produces climate simulations that are practically identical to those of the original GCM. It is noteworthy that the NN emulation developed has the same inputs and outputs as the original parameterization and is used precisely as its functional substitute within the model.

(iii) That accurate NN emulations are robust and very fast (10 to 10^5 times faster than the original parameterization) so the significant speed-up of HEM calculations can be achieved without compromising accuracy.

(iv) That statistical (NN) components can be successfully combined with deterministic model components within the HEM so their synergy can be efficiently used for environmental and climate modeling without any negative impacts on simulation quality.

(v) That this productive synergy or new combination of state-of-the-art deterministic and NN emulation approaches leads to new opportunities in using HEMs for environmental and climate simulations and prediction. For example new more sophisticated parameterizations, or even "superparameterizations" such as a cloud resolving model, that are extremely time consuming or even computationally prohibitive if used in their original form will become computationally "affordable" in ENMs when using their accurate and computationally much more efficient NN emulations in HEMs.

11.7.2 Limitations of the Current Hybrid Modeling Framework

The development of NN emulations, the core of the hybrid modeling approach, depends significantly on our ability to generate a representative training set to avoid using NNs for extrapolation far beyond the

domain covered by the training set. Because of high dimensionality of the input domain that is on the order of several hundreds or more, it is rather difficult to cover the entire domain, especially the "far corners" associated with rare events, even when we use simulated data for the NN training. Another related problem is that NN emulations are supposed to be developed for an environmental or climate system that changes in time. This means that the domain configuration for a climate simulation may evolve over time, for example, when using a future climate change scenario. In both situations described the emulating NN may be forced to extrapolate beyond its generalization ability and may lead to errors in NN outputs and result in simulation errors in the corresponding HEM. The next subsection is devoted to addressing these issues.

11.7.3 Current and Future Developments of the Hybrid Modeling Approach

Two new techniques are being developed to take care of the kind of problems outlined in the previous section and to make the NN emulation approach suitable for long-term climate change simulations and other applications – a *compound parameterization* (CP) and a NN *dynamical adjustment* (DA) (Krasnopolsky and Fox-Rabinovitz 2006a, b). Here they are only briefly outlined.

CP consists of the following three elements: the original parameterization, its NN emulation, and a quality control (QC) block. During a routine HEM simulation with CP, QC block determines (at each time step of integration at each grid point) based on some criteria whether the NN emulation or the original parameterization has to be used to generate physical parameters (parameterization outputs). When the original parameterization is used instead of the NN emulation, its inputs and outputs are saved to further adjust the NN emulation. After accumulating a sufficient number of these records, a DA of the NN emulation is produced by a short retraining using the accumulated input/output records. Thus, the adapted NN emulation becomes dynamically adjusted to the changes and/or new events/states produced by the complex environmental or climate system.

There were different possible designs considered for QC (Tolman and Krasnopolsky 2004, Krasnopolsky and Fox-Rabinovitz 2006a, b). The first and simplest QC design is based on a set of regular physical and statistical tests that are used to check the

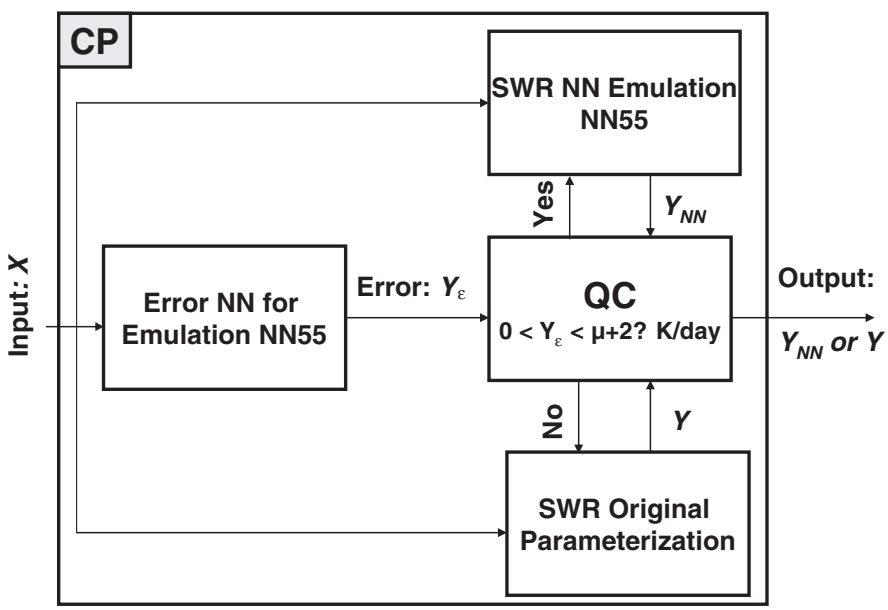


Fig. 11.5 Compound parameterization design for the NCAR CAM SWR. For each NN emulation (NN55 in this case), additional NNs (Error NN) is trained specifically for predicting, for a particular input, X, the errors, Y_ε, in the NN emulation output Y_{NN}. If these errors do not exceed a predefined threshold (mean value plus two standard deviations in this case), the SWR NN emulation (NN55) is used; otherwise, the SWR original parameterization is used instead of the NN emulation

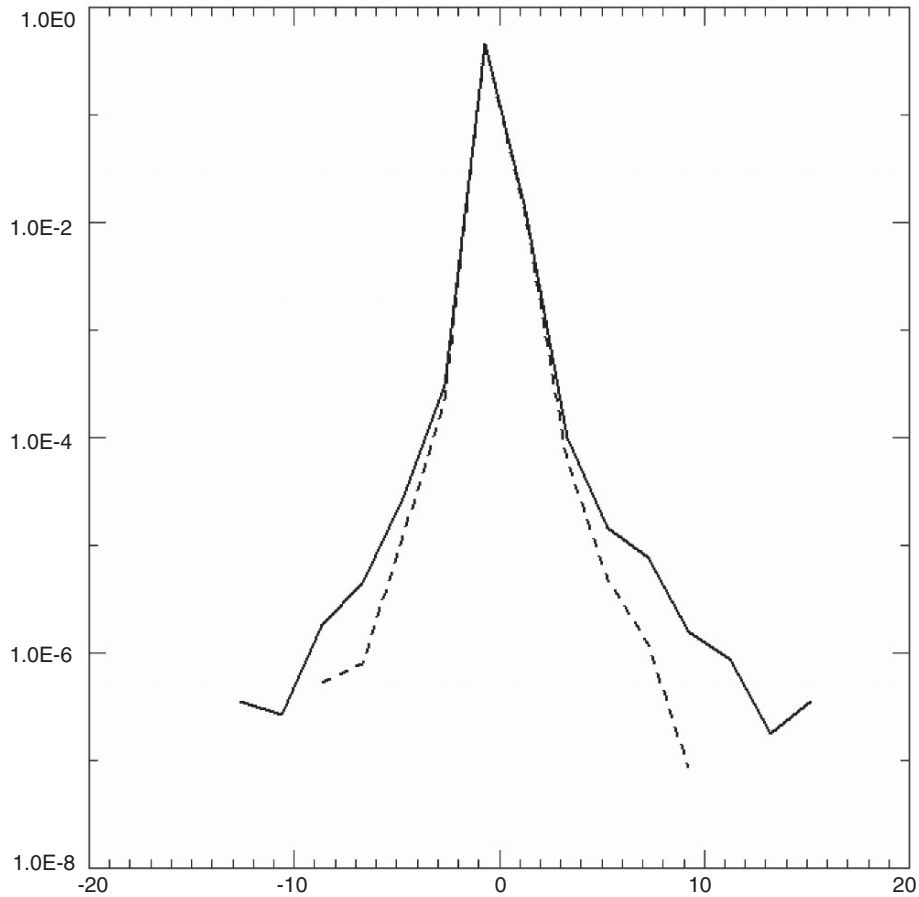


Fig. 11.6 Probability density distributions of emulation errors for the SWR NN emulation NN55 (solid line) and for the compound SWR parameterization (dashed line) are shown in Fig. 11.5. Both errors are calculated vs. the original SWR parameterization. Compound parameterization reduces the probability of medium and large errors an order of magnitude. Vertical axis is logarithmic

consistency of the NN outputs. This is the simplest, mostly generic but not sufficiently focused approach.

The second more sophisticated and effective QC design is based on training, for each NN emulation, an additional NN to specifically predict the errors in the NN emulation outputs from a particular input. If these errors do not exceed a predefined threshold the NN emulation is used; otherwise, the original parameterization is used instead. A CP of this design was successfully tested for the NCAR CAM SWR. For the SWR NN55 (see Section 11.5.3) an error NN was trained which estimated a NN55 output error *prmse(i)* (11.10) for each particular input vector X_i. The design of the CP in this case is shown in Fig. 11.5. Figure 11.6 shows the comparison of two error probability density functions. One curve (solid line) corresponds to the emulation errors of NN55, another (dashed line) corresponds to the emulation errors of the CP shown in Fig. 11.5 (both errors are calculated vs. the original parameterization on the independent test set; vertical axes are logarithmic). Figure 11.6 demonstrates the effectiveness of CP; the application of CP reduces medium and large errors by about an order of magnitude. Figure 11.7 demonstrates the effectiveness of CP in removing outliers, and Table 11.5 shows improvements in other statistical measures. It is noteworthy that for this CP less than 2% of the SWR NN emulation outputs are rejected by QC and calculated using the original SWR parameterization. Further refinement of the criteria used in the QC may result in a reduction in the already small percentage of outliers.

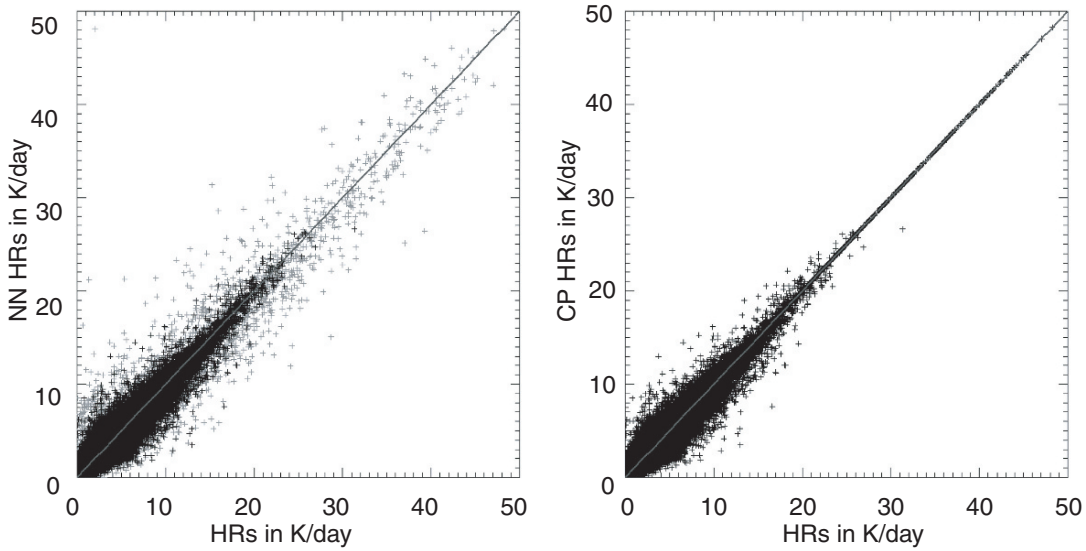


Fig. 11.7 Scatter plot for HRs calculated using the SWR NN emulation NN55 (left figure) vs. the original SWR parameterization (left and right horizontal axes) and for HRs calculated using the SWR compound parameterization (right figure) vs. the original SWR parameterization. Gray crosses (left figure) show outliers that will be eliminated by the compound parameterization (right figure)

The third QC design is based on the domain check technique proposed in the context of NN applications to satellite remote sensing (see Chapter 9, Section 9.5). In this case, QC is based on a combination of forward and inverse NNs. This design has already been successfully applied, as a preliminary study, to the ocean wave model (Section 11.6) (Tolman and Krasnopolsky 2004). Figure 11.8 illustrates the CP design in the case of the NNIA described in Section 11.6.

The parameterization Jacobian, a matrix of the first derivatives of parameterization outputs over inputs, may be useful in many cases. For example, in data assimilation applications (an optimal blending of observational and simulated data to produce the best possible fields) a Jacobian is used to create an adjoint (a tangent-linear approximation). A Jacobian is also instrumental for a statistical analysis of the original parameterization and its NN emulation. An inexpensive computation of the Jacobian when using a NN emulation is one of the advantages of the

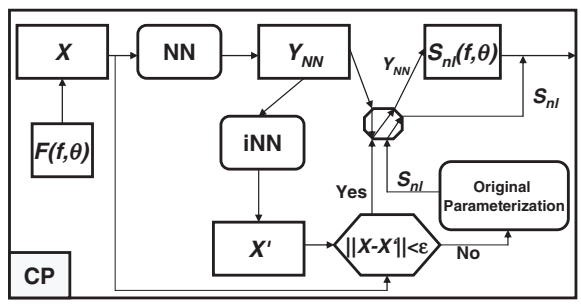


Fig. 11.8 Compound parameterization design for the NNIA and NNIAE algorithms described in Section 11.6 and shown in Fig. 11.4. Due to the use of the EOF decomposition and composition procedures the inverse NN (iNN) and QC block is implemented on the level of composition coefficients X and X'

NN approach. Using this Jacobian in combination with the tangent-linear approximation can additionally accelerate the calculations (Krasnopolsky et al. 2002). However since the Jacobian is not trained, it is simply calculated through the direct differentiation of

Table 11.5 Error statistics for SWR NN emulation NN55 and SWR compound parameterization: Bias and RMSE (25), $RMSE_{26}$ (26), and Extreme Outliers (Min Error & Max Error)

	Bias	RMSE	$RMSE_{26}$	Min error	Max error
SWR NN55	4.10^{-3}	0.193	0.434	−46.1	13.6
SWR CP	4.10^{-3}	0.171	0.302	−9.2	9.5

an emulating NN. In this case the statistical inference of a Jacobian is an ill-posed problem and it is not guaranteed that the derivatives will be sufficiently accurate.

It is noteworthy that for the type of NN applications considered in this section, the NN emulation approach that treats a parameterization of model physics as a single object offers a simple and straightforward solution that alleviates the need for calculating the NN Jacobian explicitly. The adjoint tangent-linear approximation of a parameterization (e.g., of a radiation parameterization) may be considered as an independent/new parameterization, the NN emulation approach can be applied to such a parameterization, and a separate NN emulation can be trained to emulate the adjoint. For other applications that require an explicit calculation of the NN Jacobian, several solutions have been offered and investigated (Aires et al. 1999, 2004; Krasnopolsky 2006).

References

Aires, F., Schmitt, M., Chedin, A., & Scott, N. (1999). The "weight smoothing" regularization of MLP for Jacobian stabilization. *IEEE Transactions on Neural Networks*, *10*, 1502–1510.

Aires, F., Prigent, C., & Rossow, W. B. (2004). Neural network uncertainty assessment using Bayesian statistics: A remote sensing application. *Neural Computation*, *16*, 2415–2458.

Chevallier, F., Chéruy, F., Scott, N. A., & Chedin, A. (1998). A neural network approach for a fast and accurate computation of longwave radiative budget. *Journal of Applied Meteorology*, *37*, 1385–1397.

Chevallier, F., Morcrette, J.-J., Chéruy, F., & Scott, N. A. (2000). Use of a neural-network-based longwave radiative transfer scheme in the EMCWF atmospheric model. *Quarterly Journal of Royal Meteorological Society*, *126*, 761–776.

Chou, M.-D., Suarez, M. J., Liang, X.-Z., & Yan, M. M.-H. (2001). A thermal infrared radiation parameterization for atmospheric studies. *Technical Report Series on Global Modeling and Data Assimilation*, Editor Max J. Suarez (NASA/TM-2001-104606), Vol. 19.

Collins, W. D. (2001). Parameterization of generalized cloud overlap for radiative calculations in general circulation models. *Journal of the Atmospheric Sciences*, *58*, 3224–3242.

Collins, W. D., Hackney, J. K., & Edwards, D. P. (2002). A new parameterization for infrared emission and absorption by water vapor in the national center for atmospheric research community atmosphere model. *Journal of Geophysical Research*, *107*(D22), 1–20.

Hasselmann, S., & Hasselmann, K. (1985). Computations and parameterizations of the nonlinear energy transfer in a gravity wave spectrum. Part I: A new method for efficient computations of the exact nonlinear transfer integral. *Journal of Physical Oceanography*, *15*, 1369–1377.

Hasselmann, S. et al. (1985). Computations and parameterizations of the nonlinear energy transfer in a gravity wave spectrum. Part II: Parameterization of the nonlinear transfer for application in wave models. *Journal of Physical Oceanography*, *15*, 1378–1391.

Jolliffe, I. T. (2002). *Principal component analysis* (502 pp.). New-York: Springer.

Journal of Climate (1998), *11*(6) (the special issue).

Krasnopolsky, V. (1997). A neural network-based forward model for direct assimilation of SSM/I brightness temperatures. *Technical note* (OMB contribution No. 140). NCEP/NOAA, Camp Springs, MD 20746.

Krasnopolsky, V. M. (2006). Reducing uncertainties in neural network Jacobians and improving accuracy of neural network emulations with NN ensemble approaches. *Proceedings of the IJCNN2006*, Vancouver, BC, Canada, July 16–21, 2006, pp. 9337–9344, CD-ROM; (2007), *Neural Networks*, *20*, 454–461.

Krasnopolsky, V. M., & Fox-Rabinovitz, M. S. (2006a). A new synergetic paradigm in environmental numerical modeling: Hybrid models combining deterministic and machine learning components. *Ecological Modelling*, *191*, 5–18.

Krasnopolsky, V. M., & Fox-Rabinovitz, M. S. (2006b). Complex hybrid models combining deterministic and machine learning components for numerical climate modeling and weather prediction. *Neural Networks*, *19*, 122–134.

Krasnopolsky, V. M. et al. (2000). Application of neural networks for efficient calculation of sea water density or salinity from the UNESCO equation of state. *Proceedings of the Second Conference on Artificial Intelligence, AMS*, Long Beach, CA, January 9–14, 2000, pp. 27–30.

Krasnopolsky, V. M, Chalikov, D. V., & Tolman, H. L. (2002). A neural network technique to improve computational efficiency of numerical oceanic models. *Ocean Modelling*, *4*, 363–383.

Krasnopolsky, V. M., Fox-Rabinovitz, M. S., & Chalikov, D. V. (2005a). New approach to calculation of atmospheric model physics: Accurate and fast neural network emulation of long wave radiation in a climate model. *Monthly Weather Review*, *133*, 1370–1383.

Krasnopolsky, V. M, Fox-Rabinovitz, M. S., & Chou, M.-D. (2005b). Robustness of the NN approach to emulating atmospheric long wave radiation in complex climate models. *Proceedings of the International Joint Conference on Neural Networks*, Montréal, Québec, Canada, July 31–August 4, 2005, pp. 2661–2665.

Lorenz, E. N. (1956). *Empirical* orthogonal functions and statistical weather prediction. *Statistical forecasting project* (Sci. Rep. No. 1). Cambridge, MA: MIT Press, 48 pp.

Tolman, H. L. (2002). User manual and system documentation of WAVEWATCH III version 2.22. *Technical note 222*. NOAA/NWS/NCEP/MMAB, p. 133, Camp Springs, MD 20746.

Tolman, H. L., & Krasnopolsky, V. M. (2004). Nonlinear interactions in practical wind wave models. *Proceedings of 8th International Workshop on Wave Hindcasting and Forecasting*, Turtle Bay, Hawaii, 2004, CD-ROM, E.1.

Tolman, H. L., Krasnopolsky, V. M., & Chalikov, D. V. (2005). Neural network approximations for nonlinear interactions in wind wave spectra: Direct mapping for wind seas in deep water. *Ocean Modelling*, *8*, 253–278.

Neural Network Modeling in Climate Change Studies

12

Antonello Pasini

12.1 Introduction

At present, climate change is a "hot topic", not only in scientific analyses and papers by researchers, but also in wider discussions among economists and policy-makers.

In whatever area you are, the role of modeling appears crucial in order to understand the behavior of the climate system and to grasp its complexity. Furthermore, once validated on the past, a model represents the only chance to make projections about the future behavior of the climate system.

In this framework, AI methods (more specifically, neural networks – NNs) have recently shown their usefulness in modeling studies dealing with the climate system. Thus, the aim of this paper is to review and discuss the applications of neural network modeling to climate change studies. In doing so, we will meet at least two strategies of application: the first one is "complementary" to the standard dynamical modeling *via* Global Climate Models (GCMs), due to its substantial nature of NN post-processing of GCM outputs; the second one is more "alternative" to dynamical modeling, because it is founded on the direct modeling of the climate system by NNs in an empirical and data-driven way.

Thus, in the next section we will present studies about GCMs downscaling *via* neural networks, an activity that has become quite standard in research papers during the last few years, even if not even thoroughly applied. Then, considerations about the application of dynamical modeling to a complex system like climate will lead us to recognize some weaknesses in the simulated reconstruction of the system itself. In particular, as we will see, this can lead us to criticize this kind of modeling (when applied to complex systems) and to not rely on its results.

In this unsatisfactory situation, a more phenomenological approach to the analysis of climate behavior can be applied. In Section 12.3 we will initially show how some modelers used neural network modeling for studying and forecasting the occurrence of specific phenomena, like El Niño. Then, in what follows, a comprehensive analysis of the influence of natural and anthropogenic forcings on global and regional temperature behaviors will be performed by NNs, shedding light on the most important forcings that drove the observed trends in the past.

Furthermore, in Section 12.4, a particular application of neural network modeling to the analysis of predictability on unforced and forced Lorenz attractors, which can mimic present and future climatic conditions, will be presented and discussed.

Finally, in the last section, brief conclusions will be drawn and prospects of future applications will be envisaged.

12.2 Post-processing of GCMs and Downscaling

As discussed in the first chapter of this book, the discovery of deterministic chaos in meteorological models (by Lorenz) led to reconsidering statistical methods for the forecasting activity, like MOS or Perfect Prog

Antonello Pasini (✉)
CNR – Institute of Atmospheric Pollution, Via Salaria Km 29.300, I-00016 Monterotondo Stazione (Rome), Italy
Phone: + 39 06 90672274; fax: + 39 06 90672660;
email: pasini@iia.cnr.it

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

(see Marzban et al. 2005, for a brief discussion of these methods, and for the introduction of a new method that is bias-free and shows lower uncertainty than MOS and Perfect Prog). These methods are usually applied to the post-processing of a meteorological model, too, essentially for achieving a local prediction starting from an area forecast given by the dynamical model, characterized by a finite resolution.

Recently, NNs have been used for this aim, due to their capacity for modeling nonlinear relationships between large-scale variables/patterns and local variables of fundamental importance, like temperature and precipitation. The reader can see Casaioli et al. (2003), Marzban (2003), and Yuval and Hsieh (2003) for further information and examples of applications.

Due to the fact that climate models are generally endowed with coarser resolution than those of meteorological models, one should expect that a kind of post-processing must be applied to GCMs if one wants to achieve high-resolution reconstructions or projections of climate at a regional scale. And this is exactly what happens!

12.2.1 Rationale of Downscaling

As a matter of fact, due to the very large amount of computer time needed for global simulations over several decades even on big and expensive supercomputers, the GCMs are rarely endowed with a horizontal grid spacing less than 100–150 km. This fact leads to spatially averaged reconstructions and projections that do not correctly simulate the influence of mesoscale and microscale features on regional or local climate.

The rationale for downscaling is depicted in Fig. 12.1. Here, given a certain averaged simulation result by a GCM over a grid area (bottom part of the figure), we obtain a unique value for meteo-climatic variables, like temperature and precipitation, in this area. This value should be representative of several places in the area that, on the other hand, are characterized by different physiographic features, like the presence of flat land or mountains, rivers, arid lands or forests, and human-induced changes, like land use, urbanization and the presence of industrial activities (upper part of the figure). Of course, in the past these features led to differences in parameters like the monitored surface air temperature and the amount of precipitation at different sites inside the area.

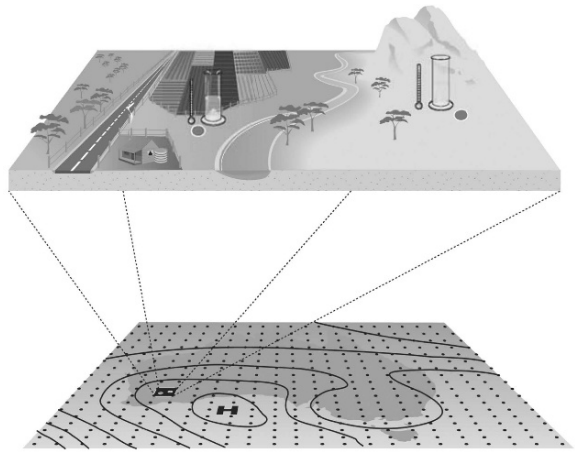


Fig. 12.1 The rationale of downscaling. Source: Australian Bureau of Meteorology, copyright Commonwealth of Australia. Reproduced by permission

Thus, the climatic reconstruction of a GCM at regional and local scales cannot be accurate; its projections for the future would be even worse. The problem of downscaling is therefore how to pass from global reconstructions and projections to regional and local ones.

12.2.2 Statistical Downscaling

Following a dynamical paradigm, the basic idea for solving this problem is to enhance resolution, either by using a purely meteorological model driven by lower boundary conditions coming from a complete GCM, or by "nesting" a complete regional climate model (RCM) in a GCM just for a limited area of the globe, with all the boundary conditions coming from the GCM and, possibly, taking the feedbacks of the regional scale on the global one into account. In either method one can run the models at high resolution to obtain a dynamical downscaling. Here we do not discuss features, results, or pros and cons of these methods: we just note that RCMs seem to guarantee a better consistency with the global picture given by GCMs, even in the absence of feedbacks from the regional scale to the global one. The interested reader can see Wang et al. (2004) for a recent review of RCMs.

From another point of view, as we have seen, the problem of climate downscaling is analogous to that of achieving local forecasts from meteorological models. Thus, it is not astonishing that statistical methods (and NNs) can play a role in this activity. As a matter of fact, at present, statistical downscaling is under active investigation in the field of climate modeling and it is performed mainly through the following techniques: regression modeling, weather classification and weather generators. Here we concentrate on regression modeling: for weather classification schemes and weather generator models, see Wilby et al. (2004).[1]

In short, statistical downscaling (and regression modeling in particular) is based on the viewpoint that the regional/local climate is conditioned by two factors: the large-scale climatic state and the regional/local physiographic features (e.g., topography, land/sea distribution, land use). So the standard process for a statistical downscaling reads as follows:

- To work out and validate a statistical model that is able to link large-scale climate variables (predictors) with regional/local variables (predictands)
- To feed the large-scale output of a GCM to the statistical model
- To estimate the corresponding regional/local climate characteristics

Given these features of statistical downscaling, it is not difficult to realize that it shows several advantages when compared with dynamical downscaling. For instance, the techniques used for building and applying the statistical model are usually quite inexpensive from the computer-time point of view, at least if compared with the long runs needed by RCMs. Furthermore, the statistical methods can be used to provide site-specific information, which can be critical for many climate change impact studies.

On the other hand, one must be aware that these statistical models suffer from a major theoretical weakness, because we are not able to verify the basic assumption that underlies them. That is to say, we cannot be sure that the statistical relationships developed for the present-day climate also hold under the different forcing conditions envisaged for the future: requiring this is a sort of "stationarity" assumption. We have to stress, however, that this is a limitation that also affects the physical parameterizations of GCMs.

What's about the choice of predictors and predictands? Of course, they must be chosen carefully. First of all, we have to consider the influence of predictors on predictand and choose the most relevant ones. Furthermore, we must be sure that predictors relevant to a regional/local predictand are adequately reproduced by the GCM to be downscaled. For instance, we know many global or regional patterns, like El Niño Southern Oscillation (ENSO) or the North Atlantic Oscillation (NAO), which are very relevant for determining the climate over particular regions: we must be sure that their course, variability and phenomenology are well reproduced by the global model. Therefore, predictors have to be chosen on the balance of their relevance to the target predictand and their accurate representation by GCMs.

Among the many regression methods available for statistical downscaling, the most important and commonly applied are: multiple linear regression, canonical correlation analysis (CCA) and NNs. Here we obviously concentrate on downscaling by NNs, though in Subsection 12.2.4 we will briefly discuss some intercomparison studies and suggest some references for further mastering.

12.2.3 Downscaling by NNs

Among the regression models, NNs appear peculiar for their characteristic feature of achieving nonlinear relationships between predictors and predictands. This feature is obviously very important in the nonlinear climate system, characterized by the many closed loops of cause-effect interactions and the relative feedbacks. Furthermore, this inherent nonlinearity of the method could become increasingly crucial when dealing with regional/local variables (predictands) which are heterogeneous and discontinuous in space and time, such as daily precipitation.

With reference to Chapter 2 of this book and to the major class of feed-forward networks described therein, we would like to stress that their architecture is just ready to incorporate predictors in input and predictands in output to build a nonlinear relationship

[1] Recently, even stochastic models, such as nonhomogeneous hidden Markov models, have been applied to a downscaling activity, namely for estimation of multi-site precipitation: see Mehrotra and Sharma (2005) and references therein, for further information on these approaches.

between them. Furthermore, if the number of hidden neurons is quite low and the phenomenon of overfitting on the training set can be controlled, we can expect to find a "law" that is realistic and not physically inadmissible such as those coming from high-order polynomial regressions.

Due to these useful features of feed-forward NNs and to their extensive use in applications, we will concentrate on the application of this particular (but very general) class of networks to GCMs downscaling, by analyzing recent scientific literature. Nevertheless, one must recognize that other kinds of networks (such as Kohonen networks) can be used fruitfully as shown at the end of this subsection.

Based on examples from a pioneering study by Trigo and Palutikof on the daily temperature downscaling at a single site in Portugal from GCM grid data (Trigo and Palutikof 1999), we can show a standard methodology for obtaining downscaled quantities *via* NNs. Then we will move to describe further developments in this field, both in the application of NNs and in the coupled choice of downscaled quantities and predictors.

We demonstrate a standard approach to downscaling temperature: as explained in the previous subsection, even this case requires an identifying the 'significant relationships between the observed large-scale atmospheric circulation and local climate, which are subsequently applied to GCM output' (Trigo and Palutikof 1999). In order to do this, one needs time series of historic temperature values at the downscaling site and reanalysis data for an area that includes the site itself. Once these data are obtained, one may choose some large-scale predictors as inputs and train the NN with synchronous temperature data from historical observed time series (targets), to derive a downscaling transfer function for the network. Once this transfer function is fixed, one can extract the same predictors from runs of a GCM and use them as input to the NN (initialization) for the same period as the reanalysis data, to obtain a downscaled present-day scenario for local temperature as output. Finally, if we initialize the NN model with GCM data pertaining to the future, we will obtain a downscaled future scenario.

As cited above, the choice of predictors is very crucial. In the work by Trigo and Palutikof (1999), as in other studies about temperature downscaling, this choice was concentrated on indices related to general atmospheric circulation. At present, particular attention is devoted to the combination of dynamical and moisture variables as predictors, especially as far as the downscaling of quantities such as precipitation, that are more sensitive to humidity values, is concerned. Table 12.1 shows the six atmospheric indices (indicated by *) chosen by Trigo and Palutikof as predictors. These indices were considered for the same day of the temperature target and for the day before; furthermore, *sin* and *cos* of the Julian day were incorporated as inputs to account for the annual cycle. Thus, a total of 14 inputs were chosen and several networks were endowed with a single output, representing maximum or minimum temperature (T_{max} or T_{min}), and a variable number of hidden neurons have been considered.

Table 12.1 The choice of circulation indices for temperature downscaling at two levels: 500 hPa and sea level pressure

Predictor	500 hPa	SLP
24 h mean (nearest grid point)	*	
24 h north-south gradient	*	*
24 h east-west gradient	*	*
24 h geostrophic vorticity		*

Source: Adapted from Trigo and Palutikof 1999. With permission from Inter-Research

The relevance of these indices for the reconstruction of local T_{max} and T_{min} has been tested by means of simple linear correlation coefficients. Recently, the use of a nonlinear correlation coefficient (the so called Correlation Ratio), that can account for nonlinear influences among variables, has been discussed for nonlinear systems to be analyzed by NNs (see, for instance, Pasini et al. 2001). Moreover, the ability of the GCM chosen to reproduce the large-scale variables has been verified for the HadCM2 model at a synoptic scale over Portugal.

Without entering into details about training and validation of the NN model,[2] it is worthwhile to focus our attention on the good reconstruction of present-day T_{max} and T_{min}, when the NN model is initialized by the large-scale indices related to the decade 1970–1979. This result is shown in Fig. 12.2a, b, where the HadCM2 reconstruction and the observed values are also shown to allow comparison. A major

[2] We just cite that usually ensemble runs of NNs with different initial random weights are performed in order to quantify the uncertainty of the final neural result. We will discuss this approach in the following section.

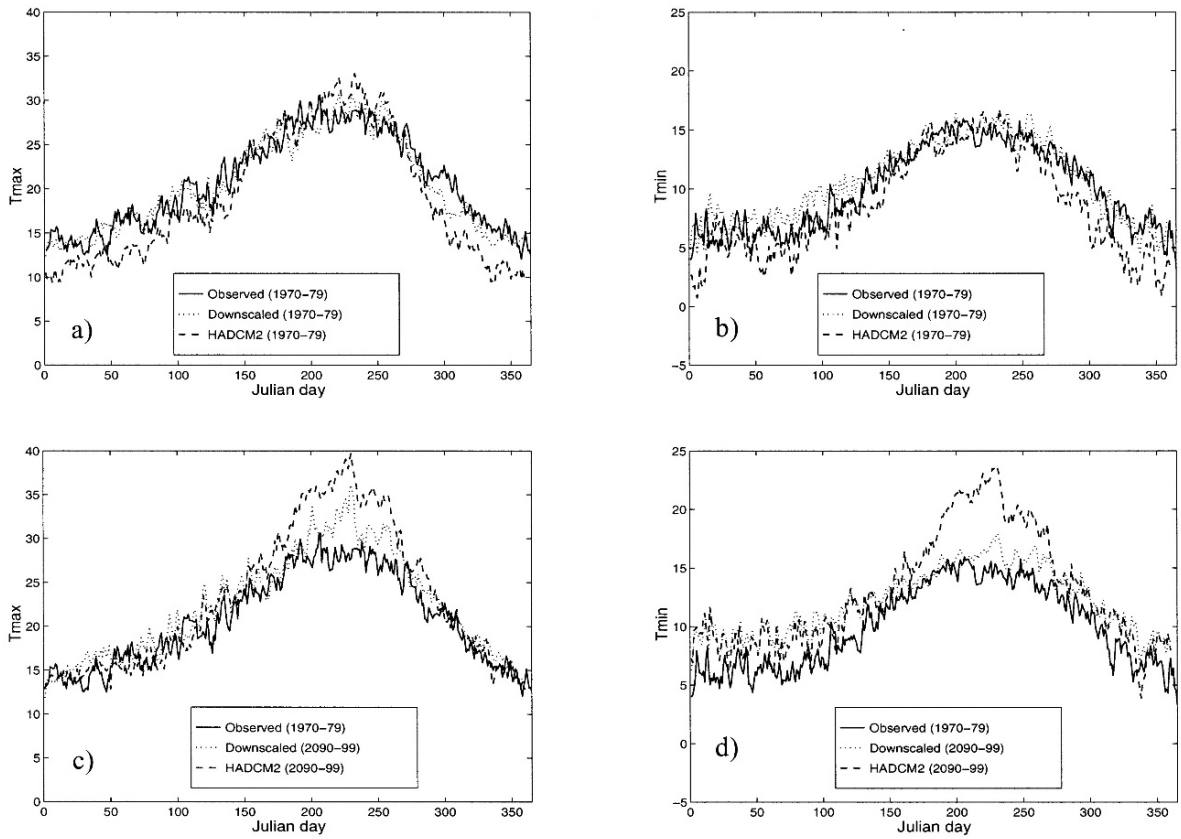


Fig. 12.2 Ten-year daily means scenarios for Coimbra (Portugal). Source: From Trigo and Palutikof 1999. With permission from Inter-Research

improvement by NN downscaling has been obtained in the cold season.

Once the downscaling model is validated on historic data, one is ready to apply the trained network to downscaling of the future climate, by feeding the network values of large-scale indices as projected into the future by the GCM. Figure 12.2c, d shows the scenarios obtained by Trigo and Palutikof for the 10-years daily averaged T_{max} and T_{min} during the period 2090–2099.

The analysis just outlined shows a standard downscaling procedure by NNs. During recent years, however, many researchers developed particular applications in this field. We would like to describe them briefly in what follows.

First of all, several studies (which can be called "pre-downscaling" ones) have been performed: in that research the ability of NNs to spatially interpolate observed atmospheric data has been shown, even in regions of complex terrain (see, for instance, Snell et al. 2000; Antonić et al. 2001). These studies further suggest using NNs for GCM downscaling if the observed atmospheric data to be interpolated (predictors) are substituted as input by GCM grid data.

As we have seen in Trigo and Palutikof (1999) standard indices of general atmospheric circulation are usually chosen as predictors for downscaling temperature with a NN. Generally, other recent studies confirm this tendency, although with some differences related to the region investigated. The situation appears more critical for precipitation downscaling. We have already cited, for example, that the importance of moisture variables or indices has been recognized in these cases. A paper that addresses this problem in a comprehensive way for daily precipitation is that by Cavazos and Hewitson (2005).

A number of papers about different climatic applications of downscaling, different types of NN architectures used and different strategies in applying them recently appeared in the scientific literature. Without

any claim of completeness, we will cite or briefly describe some of the most interesting results.

As far as the application domain is concerned, the paper by Sailor et al. (2000) is notable as an attempt at assessing the wind power implications of climate change by GCM downscaling *via* standard feed-forward NNs. In another context, the same kinds of networks have been used by Moriondo and Bindi (2006) for achieving an estimation of crop development in future climatic scenarios. In particular, those authors show the higher accuracy of NN downscaling for reconstructing present-day T_{max} and T_{min}, in comparison with GCM and RCM outputs: note that these variables are critical for the correct evaluation of extreme events that can affect crops.

Of course, one of the main fields of application for downscaling studies is hydrology. Usual multilayer perceptrons have been used in this field by Cannon and Whitfield (2002) even if with a peculiar ensemble method (known as bootstrap aggregation: see Breiman (1996)) that admits several runs of the networks and permits the decreasing of the final error in generalization performance. They downscaled large-scale atmospheric conditions from reanalysis and found the capability of NNs for correctly predicting the changes in streamflow that occurred during recent decades in a region of Canada. A comparison with stepwise linear regression has also been performed and NNs have shown the best results in this nonlinear environment.

Every study concerning rainfall variability and extremes obviously pertains to hydrological applications (and to many other impact studies). Recently, there has been a big effort to increase the performance of downscaling techniques applied to these quantities. As far as the use of NNs is concerned, we would like to cite the work by Olsson et al. 2001, where extreme rainfall has been reconstructed by standard NN downscaling, for when the rainfall amount has been divided into classes of precipitation.

More sophisticated schemes have been developed by Tatli et al. (2004) and Dibike and Coulibaly (2006). In particular, in the former paper the authors used recurrent NNs for reconstruction of the patterns of monthly total precipitation over Turkey after a preprocessing activity based on principal component analysis (PCA). In the latter paper the authors applied a time lagged feed-forward NN (TLFN) in order to downscale daily precipitation amount, T_{max} and T_{min} on a river basin in Canada. They also described the results of a comparison with a multiple linear regression method in which NNs outperform that technique for daily precipitation extremes and variability.

In a recent paper, Cannon (2006b) showed the importance of considering multivariate NNs, that is to say standard multi-layer perceptrons with more than one output, in order to account for inter-site correlations in building a pattern of a certain downscaled variable. Furthermore, he worked out a hybrid NN/analog model that performs well in multi-site climate downscaling.

Of course, as already cited, other types of networks, that do not resemble the structure of multi-layer perceptrons, are also applied to climate downscaling. The most common example is that of Kohonen networks, also called self-organizing maps (SOMs). A pioneering work on these applications is that by Cavazos (2000) who used SOMs and more standard NNs to investigate extreme events of wintertime precipitation amounts in the Balkans. In particular, SOMs have been employed to obtain climate modes and anomalies in the atmospheric patterns over the region considered, while feed-forward NNs have been used for the final downscaling activity.

Hewitson and Crane (2006) applied SOMs in a very original way, i.e. they characterized the atmospheric circulation on a quite local domain by SOMs and generated probability density functions (PDFs) for the rainfall distribution associated with each atmospheric state. For real downscaling, they took the GCM outputs, matched them to the SOM characterization of the states and, for each circulation state in the GCM data, randomly selected precipitation values from the associated PDF. They finally computed precipitation projections for South Africa, also performing a comparison of results by different downscaled GCMs.

Another "high-level" application of SOMs to clusterization and downscaling is described in the paper by Gutierrez et al. (2005). They applied this kind of network to seasonal forecasts by ensembles of models. The reader is invited to refer to their paper for technical details.

Furthermore, we stress that NNs have been recently applied to evaluating dynamical models' skill and searching for an optimal combination of GCM ensembles, finally giving projections in probabilistic terms, if possible. For the first attempts in this

direction, see Knutti et al. 2003, 2005, and Boulanger et al. 2006, 2007.

As a final remark, we note that some researchers stressed the importance of a cross-validation of the downscaling model from observational and reanalysis data for periods that represent independent or different climate regimes: for instance the last 25–30 years, characterized by an evident climate change, and the previous decades. In this way one somewhat validates the "stationarity" assumption, i.e. the most critical point of statistical downscaling techniques.

12.2.4 Intercomparison Studies

How do NNs perform in downscaling when compared with other statistical or dynamical techniques? As we have seen in the previous subsection, some papers about NNs applied to downscaling also addressed this problem, generally concluding that NNs present some advantages if compared with other methods: see, for instance, Cannon and Whitfield (2002), Moriondo and Bindi (2006), Dibike and Coulibaly (2006). Due to the importance of this topic, specific intercomparison studies have been performed recently. Here, we briefly describe them.

As clearly stated in Chapter 3 of this book, the statistical analysis of model performance is not an easy task, many indices of performance can be used and univocal conclusions can not always be drawn. Some general trends, however, in downscaling performances of different techniques can be observed.

An intercomparison study for performances in downscaling daily T_{max} and T_{min}, daily precipitation and total monthly precipitation has been performed by Schoof and Pryor (2001) in the framework of regression-based techniques of statistical modeling. After having chosen as predictors some indices of the large-scale circulation derived from PCA and cluster analysis, they tested the downscaling performance of multiple linear regression and NNs (multi-layer perceptrons). Even if differences among the various seasons are sensible, in general NNs perform better for downscaling T (especially T_{max}), for comparing correlation coefficients between predicted and observed, but also with regard to bias and to the simulation of mean and standard deviation of the predictand. Furthermore, in this study both linear and nonlinear methods substantially failed in downscaling daily precipitation, even if NNs show a quite good agreement in total monthly rainfall reconstruction.

In Trigo and Palutikof (2001) an analysis of performance in downscaling has been performed taking the "complexity" of NNs into account. The main result of this paper is that linear NNs and NNs with many hidden neurons perform worse than NNs of intermediate complexity for downscaling precipitation.

In Weichert and Bürger (1998) standard linear techniques and a radial basis function (RBF) NN are compared for downscaling temperature, precipitation and water vapor. The main result is that NN achieves the best results for reconstructing mean and variability of these quantities. In particular, heavy summer convective rainfall events, which are predominantly nonlinear, are often "detected" by the NN, while the linear method misses them completely.

In Miksovsky and Raidl (2005) three nonlinear techniques (local linear models, standard multi-layer perceptrons and RBF NNs) are compared for time series prediction and downscaling daily temperatures. No definite conclusion about performance has been drawn by this comparison for nonlinear downscaling techniques, even though all these methods perform better than multiple linear regression. Some consideration has been made of the practical application of these methods, stressing how multi-layer perceptrons have the problem of avoiding local minima, a drawback that cannot be easily handled in practice.

In Khan et al. (2006) the authors compared NNs (in particular TLFNs) and two other techniques for statistical downscaling in terms of uncertainty assessments exhibited in their downscaled results for daily precipitation and daily T_{max} and T_{min}. In general the results were achieved by calculating means and variances of downscaled quantities and by comparing them with the analogous quantities from observations. In general, even if results change month by month, NNs do not appear particularly capable of reproducing some statistical characteristics of observed data in downscaled results. In our opinion, this problem could be overcome, at least partially, if ensemble runs of NNs would be considered more extensively in the future.

Finally, in Haylock et al. (2006) a comparison of dynamical and statistical methods of downscaling has been performed for reconstruction and projection of heavy precipitation over the UK through some indices.

NNs were found to be the best technique for modeling the inter-annual variability of these indices and, by a novel approach, they also partially overcome the known problem of underestimation of extreme events. As a final note, we stress that even this work, as with many other studies regarding dynamical modeling, suggests the using ensemble runs of models in order to achieve reliable projections for future climate.

Even if there are differences among the results of the many intercomparison studies, a general conclusion is that NNs have an important role in statistical downscaling of GCMs, especially where nonlinearities among predictors and predictands are important. At present NN modeling represents an important tool for downscaling studies. Furthermore, NNs scores (and in general the scores of methods of statistical downscaling) are comparable with those coming from the "state of the art" dynamical downscaling *via* RCMs.

12.3 Neural Network Modeling as an Empirical Strategy for Climate Analysis

In the previous section we have analyzed and reviewed the most common application of NN modeling in climatic studies, i.e. downscaling. Now we will discuss other interesting and original developments of NN applications in this field.

12.3.1 Some Non-downscaling Specific Applications of NNs in Climatic Studies

First of all, we would like to remind the reader that applications of NN modeling to specific climate topics have been already presented in this book. In particular, in Chapter 10, William Hsieh described the foundations of nonlinear principal component analysis *via* NNs and showed its application to the study of phenomena of climatic relevance, like El Niño and the Quasi Biennial Oscillation in the equatorial stratosphere. Furthermore, in Chapter 11, Vladimir Krasnopolsky presented a NN use still related to dynamical climate models, such as downscaling, but now applied in a way that is very original and not dependent on model outputs. In fact, he described how NN independent modules can be substituted for physical parameterization routines in climate models and focused on the advantages of this development.

A further well founded application of NN modeling to studies with a climatic "flavor" is seasonal forecasting of sea surface temperatures (SSTs) in the tropical Pacific, which predicts the phases of El Niño Southern Oscillation (ENSO). In Wu et al. (2006) and references therein, the reader can find details of this activity and its results. Here we would like to stress that this nonlinear technique leads to forecasting improvements when compared with results from a linear method and that these improvements are especially relevant in the western equatorial Pacific region. Even if ENSO is traditionally focused on the SST variability over the central-eastern Pacific where the SST has the largest variance, the main atmospheric convective activity occurs in the western region, which is the warmest one. Thus, the possibility of improving forecasts of even small changes in the SST in this zone can lead to improvements in capturing extratropical climate behavior if predicted SSTs are used as bottom boundary conditions for an atmospheric model.

Other meteo-climatic phenomena, like monsoons, have been analyzed with the help of some kind of NNs. For instance, Cavazos et al. (2002) performed a classification of wet monsoon modes in southeast Arizona by means of SOMs, which permits studying the intraseasonal variability of that zone. Furthermore, Leloup et al. (2007) used SOMs for detecting decadal changes in ENSO and suggested the use of this methodology for studying ENSO seasonal predictability, too.

Finally, we would like to cite the interesting papers by Wu and Hsieh (2002) and by Cannon (2006a) where a nonlinear canonical correlation analysis and a nonlinear principal predictor analysis have been developed in terms of NNs and applied to the study of tropical Pacific features (in the former paper) and to the Lorenz attractor (in the latter one: as we will see in the next section, the Lorenz model is a toy model that mimics some characteristic features of the climate system).

12.3.2 Drawbacks of Dynamical Climate Modeling and the Need for an Alternative Strategy

If we briefly digress to reconsider the history of science, since from the first observations of the natural world by ancient Greek philosophers through experimental and theoretical studies on nature by modern scientists, we can recognize a big "jump" in the "style" of scientific investigations carried out by Galileo Galilei in the seventeenth century. Since Galilei adopted his experimental scientific method, more and more scientists have studied a "simplified reality" in their laboratories, for instance by extracting and investigating a single phenomenon under the influence of one or few causes, and decoupling it from other influences that are present in the real world.

In doing so, we achieved a lot of experimental data and theoretical knowledge (laws) in basic areas of investigations: at present all the so called "hard sciences" are based on this scientific praxis. In this manner, for instance, we discovered the fundamental laws of thermodynamics and fluid dynamics and, once we considered air as a mixture of gases (and of water in its three states) or as a fluid, we can apply those laws to describe its behavior.

In general, the way to obtain knowledge in less simplified and more realistic situations is also clear: one has to add further influences, one at once, in the laboratory and look at their effect on the laws previously found. Thus, for instance, we could add the influence of friction in the study of motion over a surface and find that it leads to the insertion of a further term in the equation of motion.

But, unfortunately, realistic natural systems are often composed by many subsystems and the interactions among them can be very complex, i.e. full of nonlinear cause-effect chains with feedbacks. In these situations, even if we can "decompose" the system into subsystems and study them separately in a laboratory, usually we are not able to add the influences of other subsystems and to "recompose" the complexity of the real system in experimental conditions. This is the case for both the atmosphere and the climate system.

Thus, until a few decades ago, meteorology and climatology remained observational disciplines: theoretical syntheses were obtained only with great difficulty. Recently, dynamical modeling has allowed scientists to recompose real systems in a "virtual laboratory", the computer, and to mimic their behavior by means of simulation. In these models, fundamental subsystems, phenomena and processes are described by dynamical equations (and parameterization routines for sub-grid processes) and the interactions (with feedback) among them are simulated by coupling these equations in one or more mathematical systems.[3]

Despite this great merit of dynamical models, which allow us to handle real-life complex systems, their "decomposition-recomposition" strategy suffers at least from a major drawback in climate applications. In fact, modelers are often forced to fine tune in order to establish the values of coupling parameters that lead to a correct balance of the "strength" of the many interactions/feedbacks and permit a satisfying simulation/validation of the model in reconstructing the observed climate. Even if the choice of these values is sometimes strongly driven by theoretical considerations, nevertheless the "need" for this choice makes evident the impossibility of achieving a univocal reconstruction of the simulated system in our virtual laboratory. Of course, this fact weakens our confidence in the reliability of GCM simulation results.

This drawback, together with recognizing the very simple structure of the simulated routines for several processes, leads to the search for an approach to studying the climate system that can overcome these problems, by avoiding the details of subsystems and feedbacks, that, as we have seen, cannot be reconstructed univocally by a GCM. Perhaps, it is worthwhile to have an overall look at the climate system and to analyze its behavior when considering it as the result of all the interactions and feedbacks among the various subsystems that compose it. In this way, for instance, we can search for relationships among different macroscopic variables that shed light on the climate behavior.

In this framework, in which the dynamics-based method of GCMs has been analyzed and has shown some drawbacks, an alternative data-based or more empirical technique could be applied. It will be done in the next subsection by NN modeling.

[3] A journey "from observations to simulations" through ideas and techniques of dynamical modeling in weather and climate studies is performed in Pasini (2005).

12.3.3 An Application of NN Modeling to Analysis of Forcings/Temperatures Influence in the Climate System

Usually, the strength of a dynamical approach to the study of a system is its capacity of linking causes and effects by some dynamical relationships and of "weighting" the influences of the single causes on the final effect. In this way one is able to understand which are the driving causes for a particular effect. This general property of dynamical models is obviously present also in GCMs: for instance, a cause-effect analysis of forcings on global temperature has recently been performed and presented in the IPCC third assessment report (Houghton et al. 2001, p. 710). The main result obtained by this activity is that the behavior of global temperature in the last 140 years cannot be reconstructed if anthropogenic forcings are not considered as causes of this change. Nevertheless, the complex dynamical structure of GCMs, the simplifications adopted to describe the climate system and the characteristic problem of balancing through "fine-tuned" coupling parameters weaken the reliability of cause-effect analyses in these models: at present, these results are under critic for this reason.

In this framework, an independent and overall analysis can certainly be performed by a NN, due to its characteristic features as a multivariate nonlinear regression model that can be considered as a global correlative law-finder. In particular, as we have previously seen, the structure of feed-forward networks allows us to naturally link predictors (in input) and predictands (in output). Thus, in what follows we will briefly describe such an analysis by Pasini et al. (2006). In doing so, we will stress didactical procedures and fundamental results: for more technical details, refer to that paper.

The fundamental scope of this analysis is to test the ability of simple NNs to establish nonlinear links between forcings/circulation patterns (predictors) and global or regional temperature variations (predictands) that can explain a large amount of variance. In doing so, particular attention has been devoted in order to "weight" the magnitude of different external "causes" (predictors) on a single "effect" (predictand), so that the more important influences are identified and a plausible attribution study can be performed for the recent warming.

We will perform a global and a regional case study. We consider data on annual global temperature anomalies and Central England Temperatures (CET) during extended winters as predictands. Our predictors are solar irradiance anomalies (representative of solar activity), stratospheric optical thickness at 550 nm (representative of the volcanic activity in terms of the optical properties of the lower stratosphere), global concentration of carbon dioxide (CO_2), global emissions of sulfates (SO_x), Southern Oscillation Index (SOI), related to ENSO, and the monthly NAO index.

Hereafter, we will consider solar irradiance and stratospheric optical thickness as indices of natural forcings to the climate system, while CO_2 concentration and sulfate emissions will be considered as anthropogenic forcings, i.e. influences mainly due to human activities.

A NN development environment has been created previously (Pasini and Potestà 1995) and, after successive improvements, it is now a quite complete tool for meteo-climatic applications, endowed with some original features. Our NNs are feed-forward with one hidden layer. Backpropagation rules, with the presence of both gradient descent and momentum terms, are considered for updating weights. Our transfer functions are sigmoids, in which the arguments of the exponential function are normalized with respect to the number of connections converging to a single neuron of the hidden and output layer, respectively. A discussion of this choice and its advantages, both for handling data from nonlinear systems and to avoiding overfitting problems, can be found in Pasini et al. 2001, 2003); a sketch of this normalization will be presented in the next chapter of this book. An early stopping method is also used in order to further prevent overfitting.

We stress that some specific tools are present in our development environment in order to handle historical data and to train NNs starting from quite short time series. We will see a first example of these training tools in what follows and another example in Chapter 13 of this book.

In the present application we deal with small networks, endowed with just one output, up to five inputs and four or five hidden neurons. Once trained, every NN is nothing but a function that maps input patterns (physical-chemical forcings and/or circulation indices) to values of temperature at the same time. However, due to the nonlinear nature of a NN regression, a neural

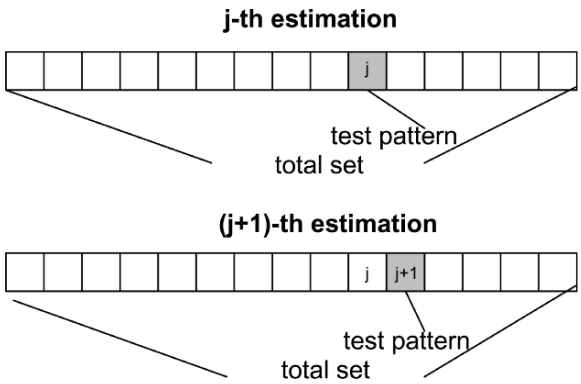


Fig. 12.3 The "all-frame" procedure. Source: From Pasini et al. 2006. With permission from Elsevier

model would be able to exactly mimic the target values without extracting any correlation law if the correspondent inputs-target patterns are included in the training set and a sufficiently large number of hidden neurons are allowed. Thus, first we consider few hidden neurons; second, we exclude some inputs-target pairs from the training set on which we build the correlative law. Once the network is trained, these pairs will be used as a validation/test set in order to assess the modeling performance on new cases that are unknown to the NN.

In this climatic application problem the statistics available are quite limited (about 140 inputs-target patterns for each case study). Therefore, we choose a facility of our tool (the so called "all-frame" procedure[4]) for estimating every temperature value at a time after the exclusion of the correspondent inputs-target pattern from the training set used for fixing the connection weights. This procedure is sketched in Fig. 12.3, where the white squares represent the patterns of the training set, while the gray square (one single pattern) represents the test set. At each step of this procedure of training + test cycles, the relative compositions of training and test sets change; a "hole" in the complete set, that represents the test set, moves across this total set of patterns, thus allowing us to estimate all the temperature values at the end of these training + test cycles.

Runs for NN and multivariate linear regression models have been performed when the models themselves were fed data about:

[4] Better known as "leave-one-out" cross-validation.

Table 12.2 Performance of reconstruction for global annual temperature

Input forcings	Linear model	Neural model
Natural	0.661	0.622 ± 0.014
Anthropogenic	0.818	0.847 ± 0.005
Nat. + anthr.	0.828	0.852 ± 0.005
Nat. + anthr. + ENSO	0.844	0.877 ± 0.004

Source: Adapted from Pasini et al. 2006. With permission from Elsevier

(a) Natural forcings only
(b) Anthropogenic forcings only
(c) Natural + anthropogenic forcings
(d) Natural + anthropogenic forcings + ENSO

Our NN analysis is exactly comparable with the dynamical analysis *via* GCM runs described in Houghton et al. (2001, p. 710). Now, data regarding ENSO are also considered because of the direct influence of this phenomenon on SST in the Pacific and its recognized teleconnections with other regions of the world.

Table 12.2 shows the performance of linear and NN models, simply assessed through the values of the linear correlation coefficient R (observed T vs. estimated T).[5] Here the uncertainty of NN results is quantified by means of ensemble runs with different initial random weights, so that each network is able to widely explore the landscape of its cost function. Each error bar indicates ± 2 standard deviations.

From these first results, we understand that NNs permit better reconstruction of temperature in the last three cases, when the NN model is able to catch nonlinearities hidden in the data and to grasp their influence in finding a nonlinear correlative law. On the other hand, the relationship between natural forcings and temperature is probably characterized by a weak nonlinear component which does not allow us to overcome the linear performance. When anthropogenic forcings are taken into account, the increase in NN performance is statistically significant (outside the error bars) when compared with the result related to the first case (natural forcings only). The further consideration of SOI index data leads to another statistically significant improvement.

A hint for interpreting these results comes from an analysis of the specific time evolution of

[5] Further assessments obtained by calculus of other indices of performance give the same trend results.

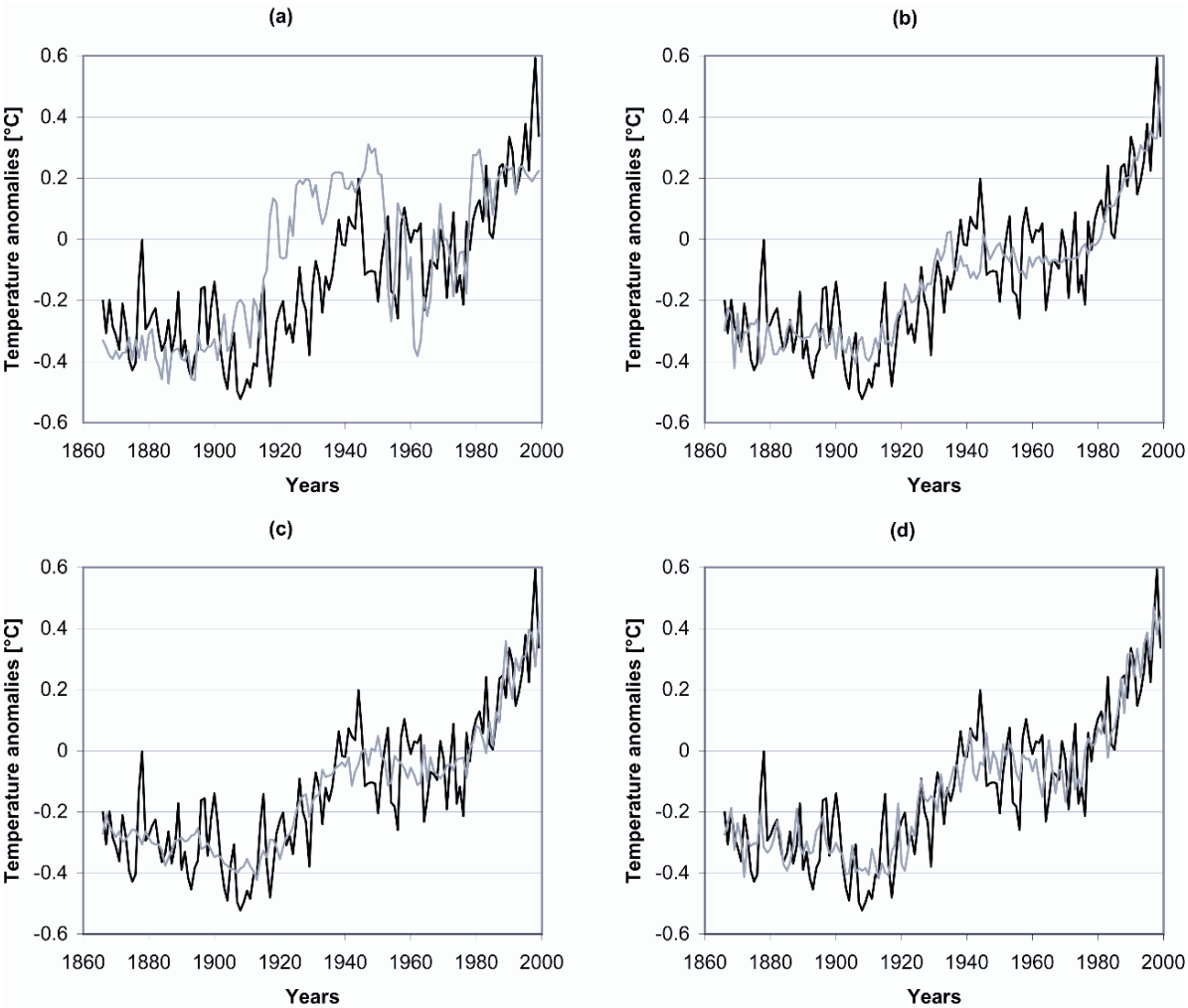


Fig. 12.4 Observed annual global temperature (black curve) vs. estimated annual global temperature by NN modeling (gray curve) when the networks are fed by inputs related to: (a) natural forcings only, (b) anthropogenic forcings only, (c) natural + anthropogenic forcings, (d) natural + anthropogenic forcings + ENSO. Source: Adapted from Pasini et al. 2006. With permission from Elsevier

reconstructed temperature by NNs versus the observed temperature record: see Fig. 12.4, where we show the results coming from a NN run for each case (a–d). The first case shows a clear failure in reconstructing the annual temperature course when just natural forcings are taken into account as inputs to the NN model. A very evident increase in performance can be appreciated in cases (b) and (c), when anthropogenic forcings are inserted as inputs. In particular, in these cases the trends are well reconstructed, in spite of an overestimation of the absolute minimum and an underestimation of the relative maximum around 1945. Furthermore, it is evident that now the amount of variance not explained by the NN model is almost completely due to the inter-annual variability of the temperature signal. Finally, when we add a further input neuron for incorporating the SOI index in the input patterns, just a better match to inter-annual variability is visible in Fig. 12.4d.

As a partial conclusion, we can say that this analysis shows the necessity of including anthropogenic forcings if we want to recover the time structure of the global temperature signal; therefore, these forcings are major influences on the temperature behavior. Moreover, ENSO clearly leads to better capturing inter-annual variability and it seems to act as a

second-order corrector to the estimation obtained by the case (c) in Fig. 12.4, even if, obviously, in a nonlinear system we cannot separate the single contributions to the final result.

As cited above, the importance of anthropogenic forcings for correctly simulating the behavior of temperature in the recent past has been recognized in dynamical studies by GCMs. The new result presented here by NN modeling confirms this "feeling". The convergence of two distinct and independent methods of considering the anthropogenic forcings as a fundamental cause that drove our climate in the last 140 years increases our confidence in the reliability of this result. Of course, this fact has important consequences for the present debate about climate change.

In Pasini et al. (2006), two other problems were analyzed: here we cite them and briefly describe the results obtained by the authors. For a deeper analysis and technical details, refer to that paper.

The amount of variance not explained by the final NN model (case (d) in Fig. 12.4) is quite low, so that one is led to ask if this amount is due to the natural variability of climate or to some hidden dynamics coming from one or more neglected dynamical causes. In order to possibly answer this question, an analysis of residuals has been performed, also by means of sophisticated techniques, such as Monte Carlo Singular Spectrum Analysis (MCSSA). Even if no undoubted conclusion can be reached by this analysis (it is well known how difficult it is to distinguish between noise and chaotic dynamical signals in short geophysical time series), we can be confident that major causes of temperature change have been considered and only 2nd-order dynamics has been neglected in this study.

Secondly, through the same strategy adopted in the global case study just described, a regional case study has been performed, analyzing the fundamental elements that drive the temperature behavior at a regional scale. Using data from Central England during extended winters (December to March), NN modeling shows that global forcings have little influence on the behavior of these temperatures, while NAO is the dominant factor for its time series reconstruction. NAO is a natural oscillation, even if at present there is a scientific debate about the possibility that its behavior can be influenced by the recent warming.

As a partial conclusion, we can affirm that a NN non-dynamical approach has aided simple assessments of the complex climate system. At a global scale, we are able to reconstruct the temperature behavior only if the anthropogenic forcings are taken into account. At a regional scale, the result obtained for Central England shows how to identify the fundamental elements that must be simulated correctly by a GCM, if one wants to achieve a correct downscaling on the zone influenced by these elements.

Finally, since identifying this phenomenological NN tool and considering its good performance in the case studies presented here, it is certainly worthwhile:

- To consider an extension to inputs related to other kinds of forcings, circulation patterns and oscillations
- To apply this NN method to other regions of the world
- To extend this treatment to the reconstruction of precipitation regimes

12.4 NN Modeling for Predictability Assessments on Unforced and Forced Lorenz Attractors

As cited in Chapter 1 of this book, the paper by Lorenz 1963, for the first time showed limits to predictability in systems that represent the atmosphere or some atmospheric phenomena, such as convection, even if in a simplified manner. This discovery contributed to opening new areas of investigation. At present, predictability is a relevant and well-developed topic in meteo-climatic studies.

The important feature of the so called Lorenz model (a simple mathematical system of nonlinear differential equations: see next subsection) is that it mimics some characteristics of both the atmosphere and the climate system, for instance their chaotic behavior and the presence of preferred states or "regimes". Furthermore, the local predictability associated with points on the Lorenz attractor resembles the predictability of single real states: for example, different points are characterized by different local Lyapunov exponents,[6]

[6] On a map or on a discrete integration of a dynamical system, the local Lyapunov exponents are the exponential rates of separation of trajectories due to a perturbation in the initial point over n steps.

just as different meteorological situations (states) are endowed with different predictabilities.[7]

In this framework, initially we will study the problem of estimating local predictability of the classical Lorenz attractor by means of NN modeling. Once we obtain satisfying results on this problem, we will add a little external forcing to the mathematical system as a toy simulation of an increase in anthropogenic forcings in the climate system, and study the related changes in predictability, still through NNs.

As usual, our treatment will be quite didactical. The reader can find more technical details in Pasini and Pelino (2005) (a first attempt at using NNs in this framework) and in Pasini (2007) (a further evolution of this approach).

12.4.1 The Lorenz Model and the Bred-Vector Growth as a Measure of Local Predictability

The Lorenz system of equations reads as follows:

$$\begin{cases} \dfrac{dx}{dt} = \sigma\,(y - x) \\ \dfrac{dy}{dt} = rx - y - xz \\ \dfrac{dz}{dt} = xy - bz \end{cases} \tag{12.1}$$

As in Lorenz (1963) in this study the values of parameters ($\sigma = 10$, $b = 8/3$, $r = 28$) are chosen in order to achieve chaotic solutions. The evolving trajectories are computed by a 4th-order Runge-Kutta integration scheme with time step $\Delta t = 0.01$.

The predictability of a state on the Lorenz attractor can be studied by considering the rate of divergence of the trajectories starting from points that are close to the state under study. Figure 12.5 graphically shows three distinct situations: note that predictability dramatically decreases when, at the middle of the attractor, trajectories undergo a "bifurcation" towards the left or the right wing of the attractor itself.

Recently, in a very interesting paper (Evans et al. 2004), the authors studied the local predictability of the Lorenz attractor by means of the so called bred-vector growth, a method which is quite similar to the computation of the local Lyapunov exponents cited above (see Kalnay et al. 2002, for a comparison of these two methods). Here, we adopt this technique for estimating local predictability.

A bred vector is a vector $\delta\vec{x}$ which simply represents the 3D-Euclidean distance between two states (points) on the Lorenz attractor after a certain number (n) of time steps in two model runs, if the second run is originated from a slight perturbation ($\delta\vec{x}_0$) in the initial conditions. We define the bred-growth rate g as:

$$g = \frac{1}{n} \ln\left(\frac{|\delta\vec{x}|}{|\delta\vec{x}_0|}\right) \tag{12.2}$$

As shown in Evans et al. (2004) g can be used to identify regions of distinct predictability of the Lorenz attractor. By adopting this method, after having fixed $n = 8$, we calculated 20,000 bred-growth rates related to points on the attractor. This allows us to distinguish zones with different predictabilities on the attractor itself and confirms in a more global manner what was anticipated in Fig. 12.5. For an overall view, see Fig. 12.6, where four classes of values of bred-growth rate are plotted at points where the related trajectories arrive on the attractor. We go from black points, where $g < 0$ and the perturbations are decaying, to the dark gray points at the bottom of the attractor and in the low parts of the wings, where a big spread ($g \geq 0.064$) between original and perturbed trajectories is evident; two further light gray classes identify intermediate (positive) values of g.

12.4.2 Is NN Forecasting Performance Related to Local Predictability?

Upon obtaining a clear estimation of local predictability of the Lorenz attractor, and considering the good performances shown by NN models in forecasting the behavior of nonlinear systems and maps, it is worthwhile to ask if there is a relationship between the scores of these performances and the values of predictability. The final idea could be: if we find a clear

[7] Predictability of a weather state is usually estimated by means of "ensemble integrations" of a model, which start from perturbations in the initial state. This operational activity shows differences in the predictability horizons of forecasting for different meteorological situations: for instance, blocking situations are generally more predictable than non-blocking ones.

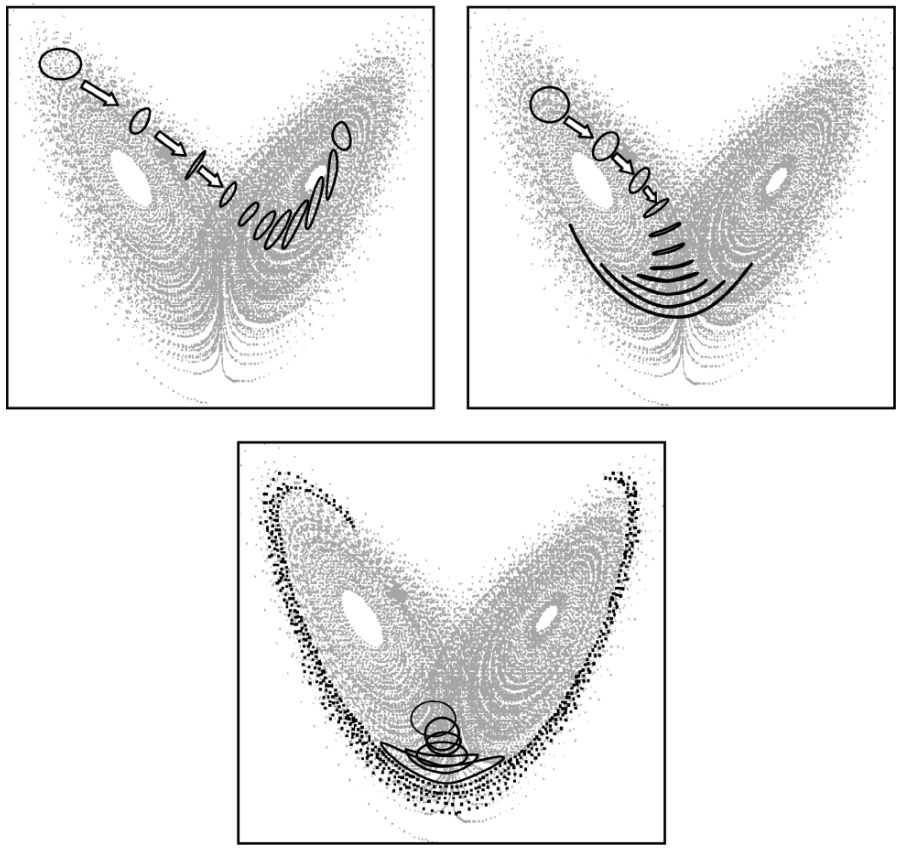


Fig. 12.5 Simulation of ensemble runs of the Lorenz model starting from perturbations in the initial states for three distinct regions of the attractor. Source: ECMWF

link between them, NN modeling can represent an alternative method to estimate predictability.

Due to different theoretical motivations (see Pasini and Pelino 2005), previous approaches to forecasting the Lorenz system essentially deal with the time series prediction of a single variable. Here, instead, we consider the full 3D dynamics of the Lorenz system and analyze the NN performance in forecasting state trajectories. In doing so, we apply the NN tool briefly described above.

By considering integrated data and fixing $n = 8$, we train our NN to make a single-step forecast from t_0 to $t_0 + 8$. The total set of Lorenz simulated data (20,000 input-target patterns) is divided into a training set (80% of data) and a validation/test set (20%). The NN topology is fixed by 3 inputs (data coordinates in the Euclidean space), 15 neurons in a single hidden layer and 3 outputs (the 3D position after 8 time steps of integration). The number of hidden neurons is sufficient to obtain a good representation of the underlying function, but not too big to overfit the data. Finally, the 3D-Euclidean distances between output and target points represent the errors made by the NN in forecasting: the mean distance error for each class of bred-growth rate can be therefore considered as a global measure of NN forecast performance.

NN performance on the test set are shown in Table 12.3. Even in this application, as in Section 12.3 for the forcings/temperatures analysis, the error bars associated with the performance measure come from

Table 12.3 Performance of NN forecasts on the test set in terms of mean distance error

Bred-growth rate	Mean distance error in NN forecasts
$g < 0$	5.66 ± 0.15
$0 \leq g < 0.04$	6.58 ± 0.25
$0.04 \leq g < 0.064$	6.62 ± 0.24
$g \geq 0.064$	8.36 ± 0.41

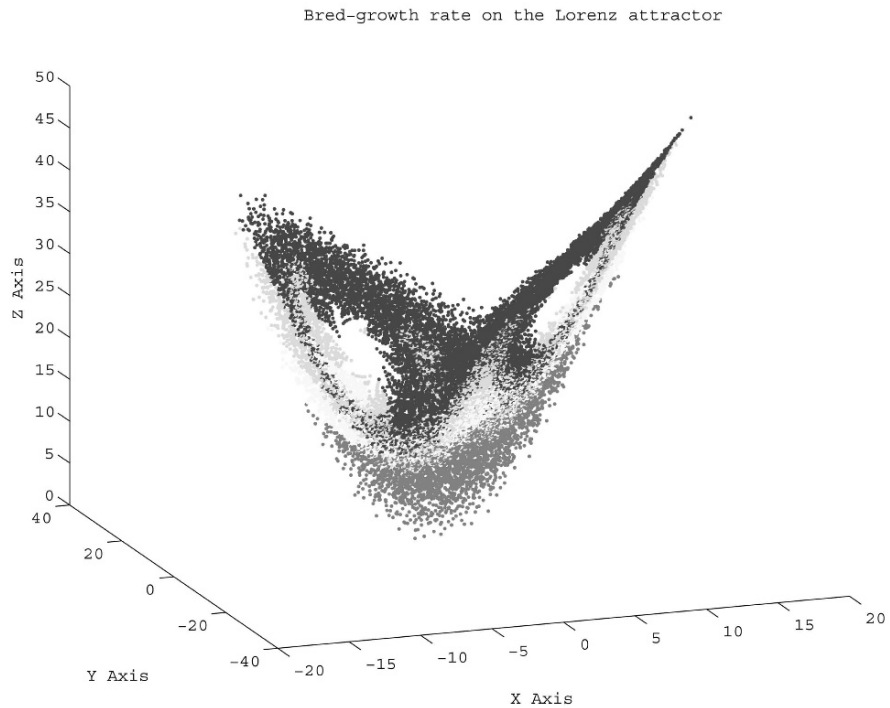


Fig. 12.6 Regions of different predictability of the Lorenz attractor estimated through classes of bred-growth rate

ensemble runs of the model with different initial random weights and represent ± 2 standard deviations. It is evident that the best performance is achieved for negative values of g and the worst one for its largest values; intermediate performance is obtained for the other two classes, which cannot be distinguished from each other by NN forecast errors but are well separated (outside the error bars) from the former two classes.

For each single run of the NN model, we can analyze the frequency distributions of distance errors for each class defined above. In this way, a clear shift from unimodal distributions (for $g < 0.04$) to quasi-bimodal distributions (for $g \geq 0.04$) can be appreciated. An example of these distributions is shown in Fig. 12.7. This fact further shows how NN forecast performance is sensitive to regions where a change of regime is possible and two close trajectories could evolve to opposite wings of the attractor. In this case, a secondary maximum is present in the distribution for large errors, when the NN forecasts a trajectory to arrive onto the wrong wing.

Thus, to our knowledge, a clear relationship between local predictability in some regions of the Lorenz attractor and the forecasting performance of a NN model has been established for the first time. This result shows that NNs can be used to identify these regions and this is quite important. However, we have to stress that this identification is obtained here by comparing NN forecasting results with "observed" targets (the final points of the integrated trajectories), while in an operational estimation of predictability we would like to assess it for future times, when observations are still not available. Then, how should we proceed in order to study the feasibility of a fully operational estimation of predictability for the Lorenz attractor?

In this didactical chapter we will not deal with this problem and its possible solutions. However, we stress that some hints for achieving these solutions could be inferred by our analysis: for instance, with reference to Table 12.3, we note that there is an increase in the amplitude of the error bars associated with NN performance when going from low bred-growth rates to high ones. This increased spread in the ensemble runs of the NN model can be related to the more complex structure of the cost function for the classes characterized by "bifurcations" and could lead to an "a priori" NN

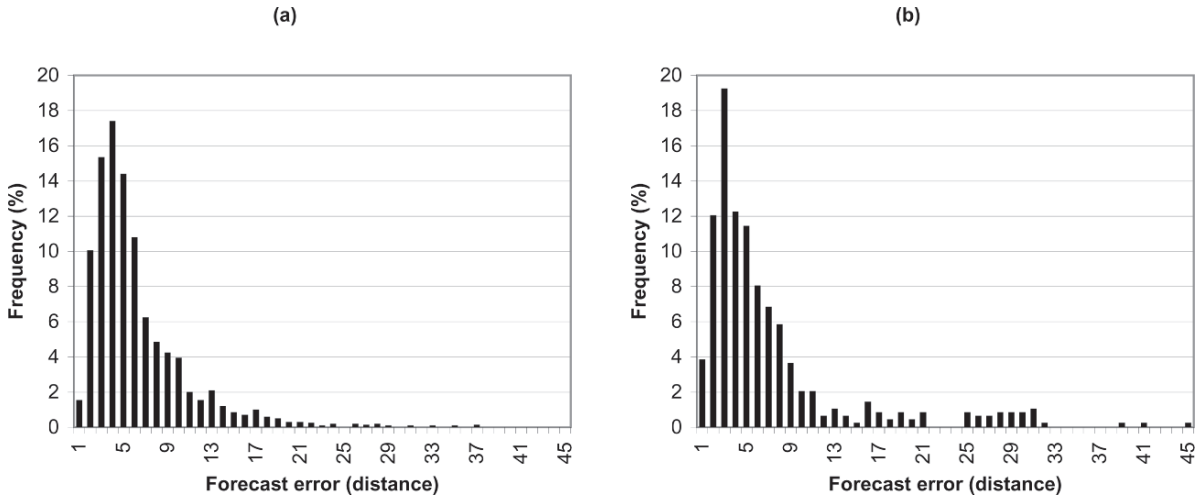


Fig. 12.7 Error distributions of distance between target and output coordinates for the two classes of bred-growth rate $g < 0$ (a) and $0.04 \leq g < 0.064$ (b). Source: Adapted from Pasini and Pelino 2005. Copyright IEEE

estimation of predictability for the regions themselves. Furthermore, as far as the predictability to be associated to single points of the attractor is concerned, one can even think of adopting a more classical "ensemble" strategy of a dynamical origin: i.e., one could start from several perturbed initial conditions and run a NN model several times to estimate a final distribution of arrival points. The spread of this distribution could be a measure of the predictability of the initial state.

Another approach is also possible: we can perform a NN forecast of bred-growth rates for any point on the attractor. This leads us to obtain a direct estimation of this variable, which well represents the local predictability values. The reader can refer to Pasini (2007) for developments of this latter approach.

12.4.3 A Forced Lorenz System and Changes in Predictability over Its Attractor

If the Lorenz model can mimic some features of the atmosphere and the climate system, is it able to answer to changes in external forcings in a way that resembles the real systems to real forcings? In recent papers (Palmer (1999); Corti et al. 1999), the authors noted that the climate change of the last decades can also be observed not only in changes of the mean relevant meteo-climatic quantities, but it can be interpreted 'in terms of changes in the frequency of occurrence of natural atmospheric circulation regimes' (Corti et al. 1999). In this framework, they found that a weak external forcing added to the classical Lorenz system leads to changes in the total persistence time of states in the two regimes (wings), i.e. in the frequency of occurrence of its regimes. This weak external forcing could be a toy simulation of the recent increase of anthropogenic forcings in the real system.

This interesting behavior of a forced Lorenz model and the fact that, to our knowledge, no paper about its predictability has appeared in the scientific literature, leads us to apply the previous (dynamical and NN) methods to its study.

A simple forced Lorenz system can be written as follows:

$$\begin{cases} \dfrac{dx}{dt} = \sigma\,(y - x) + f_0 \cos\theta \\ \dfrac{dy}{dt} = rx - y - xz + f_0 \sin\theta \\ \dfrac{dz}{dt} = xy - bz \end{cases} \tag{12.3}$$

Following an example by Palmer and coworkers (Corti et al. 1999), we choose $f_0 = 2.5$ and $\theta = 90°$ for the new free parameters of this model.

The integration of the forced system and the calculation of bred-growth rates for 20,000 trajectories

Table 12.4 Comparison of mean distance errors of the NN model in the forecasting activity on unforced and forced Lorenz attractors

Bred-growth rate	Mean distance error (unforced)	Mean distance error (forced)
$g < 0$	5.66 ± 0.15	5.41 ± 0.10
$0 \leq g < 0.04$	6.58 ± 0.25	6.66 ± 0.29
$0.04 \leq g < 0.064$	6.62 ± 0.24	6.59 ± 0.27
$g \geq 0.064$	8.36 ± 0.41	8.15 ± 0.42

allows us to obtain a picture that is very similar to that shown in Fig. 12.6. This means that the shape of the attractor is not deformed substantially and that just the frequency of occurrence of its regimes changes. An increase (about 1%) in the total number of cases with $g < 0$ is found and, at the same time, a 2% decrease of the less numerous cases with $g \geq 0.04$ can be observed. This suggests a global increase of predictability for the new attractor.

By applying the NN model to a forecasting activity in this forced case, we can establish that the global average forecast error decreases slightly. In particular, as shown in Table 12.4, a statistically significant forecast improvement is shown for the class of points characterized by decaying perturbations ($g < 0$), while for the most critical class endowed with $g \geq 0.064$ a little increase in performance is found, but not statistically significant. The NN performance on the other two classes is very similar to that of the unforced case study. Moreover, even now some kind of bimodality in the frequency distribution of forecast errors is present for classes characterized by $g \geq 0.04$.

Thus, in this forced situation a global increase in predictability is shown by both a dynamical and a NN analysis. However, the slight difference in average values of bred-growth rate and NN error does not permit attributing this change to a more frequent permanence of the system's state in regions of high predictability, or to a modification in local predictability of single points, or, finally, to both of these phenomena. A deeper dynamical analysis (Pasini and Pelino 2005) shows that this is due to the very similar dynamical properties of the unforced and the forced systems considered here, suggesting an increase in the value of the external forcing, for instance by considering $f_0 = 5$. This allows us to obtain distinct dynamical conditions and, meanwhile, to preserve a clear chaotic behavior.

In this framework of a doubled forcing, predictability increases and, analogously, NN performance increases as well. This result clearly shows the sensitivity of NN forecasting to the values of local predictability and leads to rediscovery of an increase of predictability in forced situations by a non-dynamical method. Furthermore, when an operational forecast of local predictability (represented by the value of bred-growth rate) is performed, we recognize that it is better forecasted in a forced situation (Pasini 2007).

The analysis performed here obviously represents a preliminary attempt at NN predictability estimation on a toy model like the Lorenz system. Several improvements to this approach can be made: refer to Pasini (2007) for actual developments and discussion about further prospects.

12.5 Conclusions and Prospects

Modeling the complex dynamics of the climate system is a very difficult task. In this framework, we have shown that NN modeling begins to help in grasping this complexity and in shedding light on climate behavior, both as a complementary technique that may be used together with GCMs and as an empirical alternative strategy to dynamical modeling.

In this paper, characteristic features and good qualities of NNs, original approaches and application results have been presented in this particular domain. This has allowed us to understand that, at present, NN modeling represents a helpful tool for facing climate change challenges.

Obviously, if downscaling by NN modeling is quite well established and the related developments of NN techniques and application methods have been extensively studied, the alternative strategy presented in Sections 12.3 and 12.4 is taking the first steps. Just for this reason, prospects of future development are particularly exciting in this framework, where new domains of application will certainly be opened in the future. Furthermore, the complexity of climatic problems will likely require testing new application methods and NN models endowed with more sophisticated structure and learning behavior.

If some reader is stimulated to contribute to further original applications of NNs to climate change studies, the aim of this chapter will be achieved.

References

Antonić, O., Križan, J., Marki, A., & Bukovec, D. (2001). Spatio-temporal interpolation of climatic variables over region of complex terrain using neural networks. *Ecological Modelling, 138*, 255–263.

Boulanger, J.-P., Martinez, F., & Segura, E. C. (2006). Projection of future climate change conditions using IPCC simulations, neural networks and Bayesian statistics. Part I: Temperature mean state and seasonal cycle in South America. *Climate Dynamics, 27*, 233–259.

Boulanger, J.-P., Martinez, F., & Segura, E. C. (2007). Projection of future climate change conditions using IPCC simulations, neural networks and Bayesian statistics. Part II: Precipitation mean state and seasonal cycle in South America. *Climate Dynamics, 28*, 255–271.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*, 123–140.

Cannon, A. J. (2006a). Nonlinear principal predictor analysis: Application to the Lorenz system. *Journal of Climate, 19*, 579–589.

Cannon, A. J. (2006b). A hybrid neural network/analog model for climate downscaling. *Proceedings of the 2006 IEEE World Conference on Computational Intelligence*. IEEE: Vancouver, Canada.

Cannon, A. J., & Whitfield, P. H. (2002). Downscaling recent streamflow conditions in British Columbia, Canada using ensemble neural network models. *Journal of Hydrology, 259*, 136–151.

Casaioli, M., Mantovani, R., Proietti Scorzoni, F., Puca, S., Speranza, A., & Tirozzi, B. (2003). Linear and nonlinear postprocessing of numerical forecasted surface temperature. *Nonlinear Processes in Geophysics, 10*, 373–383.

Cavazos, T. (2000). Using self-organizing maps to investigate extreme climate events: An application to wintertime precipitation in the Balkans. *Journal of Climate, 13*, 1718–1732.

Cavazos, T., & Hewitson, B. C. (2005). Performance of NCEP-NCAR reanalysis variables in statistical downscaling of daily precipitation. *Climate Research, 28*, 95–107.

Cavazos, T., Comrie, A. C., & Liverman, D. M. (2002). Intraseasonal variability associated with Wet Monsoons in Southeast Arizona. *Journal of Climate, 15*, 2477–2490.

Corti, S., Molteni, F., & Palmer, T. N. (1999). Signature of recent climate change in frequencies of natural atmospheric circulation regimes. *Nature, 398*, 799–802.

Dibike, Y. B., & Coulibaly, P. (2006). Temporal neural networks for downscaling climate variability and extremes. *Neural Networks, 19*, 135–144.

Evans, E., Bhatti, N., Kinney, J., Pann, L., Peña, M., Yang, S.-C., Kalnay, E., & Hansen, J. (2004). RISE undergraduates find that regime changes in Lorenz's model are predictable. *Bulletin of the American Meteorological Society, 85*, 520–524.

Gutierrez, J. M., Cano R., Cofiño, A. S., & Sordo, C. (2005). Analysis and downscaling multi-model seasonal forecast in Peru using self-organizing maps. *Tellus, 57A*, 435–447.

Haylock, M. R., Cawley, G. C., Harpham, C., Wilby, R. L., & Goodess, C. M. (2006). Downscaling heavy precipitation over the United Kingdom: A comparison of dynamical and statistical methods and their future scenarios. *International Journal of Climatology, 26*, 1397–1415.

Hewitson, B. C., & Crane, R. G. (2006). Consensus between GCM climate change projections with empirical downscaling: Precipitation downscaling over South Africa. *International Journal of Climatology, 26*, 1315–1337.

Houghton, J. T., Ding, Y., Griggs, D. J., Noguer, M., van der Linden, P. J., Dai, X., Maskell, K., & Johnson, C. A. (Eds.). (2001). *Climate change 2001: The scientific basis* (pp. 881). Cambridge: Cambridge University Press.

Kalnay, E., Corazza, M., & Cai, M. (2002). Are bred vectors the same as Lyapunov vectors?. *Proceedings of the Symposium on Observations, Data Assimilations, and Probabilistic Prediction, 82nd Annual Meeting of the American Meteorological Society*. AMS, CD ROM: Orlando, FL.

Khan, M. S., Coulibaly, P., & Dibike, Y. (2006). Uncertainty analysis of statistical downscaling methods. *Journal of Hydrology, 319*, 357–382.

Knutti, R., Stocker, T. F., Joos, F., & Plattner, G.-K. (2003). Probabilistic climate change projections using neural networks. *Climate Dynamics, 21*, 257–272.

Knutti, R., Joos, F., Müller, S. A., Plattner, G.-K., & Stocker, T. F. (2005). Probabilistic climate change projections for CO2 stabilization profiles, *Geophysical Research Letters, 32*(L20707). DOI: 10.1029.2005GL023294.

Leloup, J. A., Lachkar, Z., Boulanger, J.-P., & Thiria, S. (2007). Detecting decadal changes in ENSO using neural networks. *Climate Dynamics, 28*, 147–162.

Lorenz, E. N. (1963). Deterministic non-periodic flow. *Journal of Atmospheric Sciences, 20*, 130–141.

Marzban, C. (2003). A neural network for post-processing model output: ARPS. *Monthly Weather Review, 131*, 1103–1111.

Marzban, C., Sandgathe, S., & Kalnay, E. (2005). MOS, Perfect Prog, and Reanalysis Data. *Monthly Weather Review, 134*, 657–663.

Mehrotra, R., & Sharma, A. (2005). A nonparametric nonhomogeneous hidden Markov model for downscaling of multisite daily rainfall occurrences. *Journal of Geophysical Research, 110*(D16108). DOI: 10.1029/2004JD005677.

Miksovsky, J., & Raidl, A. (2005). Testing the performance of three nonlinear methods of time series analysis for prediction and downscaling of European daily temperatures. *Nonlinear Processes in Geophysics, 12*, 979–991.

Moriondo, M., & Bindi, M. (2006). Comparisons of temperatures simulated by GCMs, RCMs and statistical downscaling: Potential application in studies of future crop development. *Climate Research, 30*, 149–160.

Olsson, J., Uvo, C. B., & Jinno, K. (2001). Statistical atmospheric downscaling of short-term extreme rainfall by neural networks. *Physics and Chemistry of the Earth (B), 26*, 695–700.

Palmer, T. N. (1999). A nonlinear dynamical perspective on climate prediction. *Journal of Climate, 12*, 575–591.

Pasini, A. (2005). *From observations to simulations. A conceptual introduction to weather and climate modelling* (201 pp.). Singapore: World Scientific.

Pasini, A. (2007). Predictability in past and future climate conditions: a preliminary analysis by neural networks using unforced and forced Lorenz systems as toy models.

Proceedings of the 5th Conference on Artificial Intelligence and its Applications to Environmental Sciences, 87th Annual Meeting of the American Meteorological Society. AMS, CD ROM: San Antonio, TX.

Pasini, A., & Pelino, V. (2005). Can we estimate atmospheric predictability by performance of neural network forecasting? The toy case studies of unforced and forced Lorenz Models. *Proceedings of the IEEE International Conference on Computational Intelligence for Measurement Systems and Applications* (pp. 69–74). IEEE: Giardini Naxos, Italy.

Pasini, A., & Potestà, S. (1995). Short-range visibility forecast by means of neural-network modelling: A case study. *Nuovo Cimento, C24*, 505–516.

Pasini, A., Pelino, V., & Potestà, S. (2001). A neural network model for visibility nowcasting from surface observations: Results and sensitivity to physical input variables. *Journal of Geophysical Research, 106*(D14), 14951–14959.

Pasini, A., Ameli, F., & Lorè, M. (2003). Short range forecast of atmospheric radon concentration and stable layer depth by neural network modelling. *Proceedings of the IEEE International Symposium on Computational Intelligence for Measurement Systems and Applications* (pp. 85–90). IEEE: Lugano, Switzerland.

Pasini, A., Lorè, M., & Ameli, F. (2006). Neural network modelling for the analysis of forcings/temperatures relationships at different scales in the climate system. *Ecological Modelling, 191*, 58–67.

Sailor, D. J., Hu, T., Li, X., & Rosen, J. N. (2000). A neural network approach to local downscaling of GCM output for assessing wind power implications of climate change. *Renewable Energy, 19*, 359–378.

Schoof, J. T., & Pryor, S. C. (2001). Downscaling temperature and precipitation: A comparison of regression-based methods and artificial neural networks. *International Journal of Climatology, 21*, 773–790.

Snell, S. E., Gopal, S., & Kaufmann, R. K. (2000). Spatial interpolation of surface air temperatures using artificial neural networks: Evaluating their use for downscaling GCMs. *Journal of Climate, 13*, 886–895.

Tatli, H., Dalfes, H. N., & Mentes, S. S. (2004). A statistical downscaling method for monthly total precipitation over Turkey. *International Journal of Climatology, 24*, 161–180.

Trigo, R. M., & Palutikof, J. P. (1999). Simulation of daily temperatures for climate change scenarios over Portugal: a neural network model approach. *Climate Research, 13*, 45–59.

Trigo, R. M., & Palutikof, J. P. (2001). Precipitation scenarios over Iberia: A comparison between direct GCM output and different downscaling techniques. *Journal of Climate, 14*, 4422–4446.

Wang, Y, Leung, L. R., McGregor, J. L., Lee, D.-K., Wang, W.-C., Ding, Y., & Kimura, F. (2004). Regional climate modelling: Progress, challenges, and prospects. *Journal of the Meteorological Society of Japan, 82*, 1599–1628.

Weichert, A., & Bürger, G. (1998). Linear versus nonlinear techniques in downscaling. *Climate Research, 10*, 83–93.

Wilby, R. L., Charles, S. P., Zorita, E., Timbal, B., Whetton, P., & Mearns, L. O. (2004). *Guidelines for use of climate scenarios developed from statistical downscaling methods* (Report of the IPCC Task Group TGICA), from http://ipcc-ddc.cru.uea.ac.uk/guidelines/StatDown_Guide.pdf.

Wu, A., & Hsieh, W. W. (2002). Nonlinear canonical correlation analysis of the tropical Pacific wind stress and sea surface temperature. *Climate Dynamics, 19*, 713–722.

Wu, A., Hsieh, W. W., & Tang, B. (2006). Neural network forecasts of the tropical Pacific sea surface temperatures. *Neural Networks, 19*, 145–154.

Yuval & Hsieh, W. W. (2003). An adaptive nonlinear MOS scheme for precipitation forecasts using neural networks. *Weather and Forecasting, 18*, 303–310.

Neural Networks for Characterization and Forecasting in the Boundary Layer *via* Radon Data

13

Antonello Pasini

13.1 Introduction

The complexity of air-pollution physical-chemical processes in the boundary layer (BL) is well known: see, for instance, Stull (1988) and Seinfeld and Pandis (1998). In this framework, we do not make any attempt at reviewing the manifold use of neural networks (NNs) for air-pollution assessments and forecasting. Instead, we focus just on the (complex) physics of the BL and discuss the coupled use of an original index of the BL properties (radon concentration) and of NN modeling in order to obtain interesting results for characterizing and/or forecasting important variables in the BL, like the concentration of a dangerous primary pollutant (benzene) and the 2-h evolution of stable layer depth. In this scenario, the particular strategies for applying a NN model are described, showing how they lead to important original results, for grasping the BL physical behavior. In doing so, one can discover the usefulness of an empirical AI data-driven approach to investigating a complex system that is very difficult to deal with in terms of dynamical models.

In the next section, a brief introduction to fundamentals of radon detection will be presented and the qualitative and quantitative relevance of radon concentration for summarizing the physical state of the BL will be discussed. In particular, we will present the structure of a box model (based on radon data) for estimating the nocturnal stable layer depth.

In Section 13.3, after having introduced simple linear indices that characterize days and nights in terms of meteorological predisposition to a primary pollution event, generalized nonlinear indices are built up by a NN model: they show the ability of NNs at capturing nonlinearities in the data and achieving better modeling results.

In Section 13.4, by applying NN modeling and a particular preprocessing activity, we will show how we are able to achieve reliable short-range forecasts of radon concentration and stable layer depth in an urban environment. These forecasts represent fundamental information for assessing dispersion properties and their related influence on pollutant concentrations.

Finally, in the last section, brief conclusions will be drawn and prospects of future developments in this field will be envisaged.

13.2 Relevance of Radon in Studies of the Boundary Layer

As well known, radon is an important factor that can lead to lung cancer (see www.epa.gov for further information). Due to this fact, many epidemiological studies on radon have been performed during the last decades, especially in indoor environments. Less known is the role of radon as a "tracer" of the physical characteristic features of the lower layers of the atmosphere.

Despite this fact, it is worthwhile to stress that the first studies on the role of radon in characterizing BL

Antonello Pasini (✉)
CNR – Institute of Atmospheric Pollution, Via Salaria Km 29.300, I-00016
Monterotondo Stazione (Rome), Italy
Phone: + 39 06 90672274; fax: + 39 06 90672660;
email: pasini@iia.cnr.it

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

dispersion properties dated back to the late 1970s, when French researchers began to consider this noble gas, which undergoes no chemical reaction, as a perfect tracer of the BL dilution features. They found that counts of beta radioactivity, coming from the decay of short-lived radon daughters, represent a simple index of the stability state of the BL (see Guedalia et al. 1980, and references therein).

In general, the search for an index which is able to summarize the characteristic features of a system is seen merely as a simplification (sometimes used for working out a conceptual model of the behavior of the system) that is unnecessary and can be overcome by a dynamical description of the system itself. Nevertheless, in a highly nonlinear system, the knowledge of a suitable index whose time development mimics the behavior of some intrinsic property of the system itself gives us key information, certainly in a qualitative way and, hopefully, also in a more quantitative manner.

13.2.1 Semi-quantitative Information by Estimating an Equivalent Mixing Height

Without discussing the instrumentation for detecting beta counts from the decay of short-lived radon progeny, we just stress that the fraction attached to particulate matter is usually detected (see Allegrini et al. 1994, for details). As shown in Fig. 13.1, the typical time patterns of beta counts are maxima during the night in conditions of nocturnal stability and minima during the day when the mixed layer is well developed (the more enhanced is the stability of the nocturnal stable layer, then the higher are the maxima in radioactivity counts). Otherwise, low quasi-constant values are found in advective situations characterized by mechanical turbulence. This qualitative analysis suggests that, in general, the number of beta counts can be inversely proportional to the "degree" of stability of the lower layers.

Especially interesting cases are nocturnal stable situations over towns, when anticyclones at synoptic scale and local physiographic and emissive features can create conditions for the development of strong stable layers and peak events of primary pollution. In these cases, rain is absent, relative humidity and pressure are quite constant and the spacetime interval to be analyzed is limited (a night and a town). Thus, if we limit our study to these situations, then radon exhalation from the ground can be considered constant in time and spatially homogeneous, the attached fraction of radon daughters also constant and the radon concentration directly proportional to the number of beta counts detected.

Further evidence (see Lopez et al. 1974, for pioneering work and Vinod Kumar et al. 1999, for more recent results) shows that radon and radon daughters' concentrations are approximately homogeneous with altitude in the nocturnal stable layer and that they undergo a rapid transition to background values above the mixing height in the so-called residual layer. This fact induced Guedalia et al. (1980) to use a box model of the nocturnal stable layer endowed with a homogeneous radon concentration in the vertical. The top of this box, named equivalent mixing height h_e, has been found to be a good index of the dispersion properties of this layer in a semi-quantitative way, because low (high) values of h_e are related to low (high) dispersion power and high (low) concentrations of primary pollutants.

In Guedalia et al. (1980) the calculation of the top of the box has been performed by means of the following equation:

$$h_e(t) = \frac{\Phi \Delta t}{C(t) - C_0} \tag{13.1}$$

Here Φ is the radon flux at the surface, Δt is the time interval from the start of accumulation, $C(t)$ is the radon concentration at time t and C_0 is the radon concentration at the beginning of accumulation (evening).

13.2.2 A Physical Interpretation of the Equivalent Mixing Height

Guedalia et al. (1980) wondered what h_e should represent from a physical point of view. They correctly asserted that it is different from the inversion layer thickness, and that instead, it should be identified with the base of the inversion, although even this option is not the precise one (in particular, for ground-based inversions, the box model is no longer applicable).

The correct solution to this puzzle has been given by Allegrini et al. (1994). They show that h_e can be

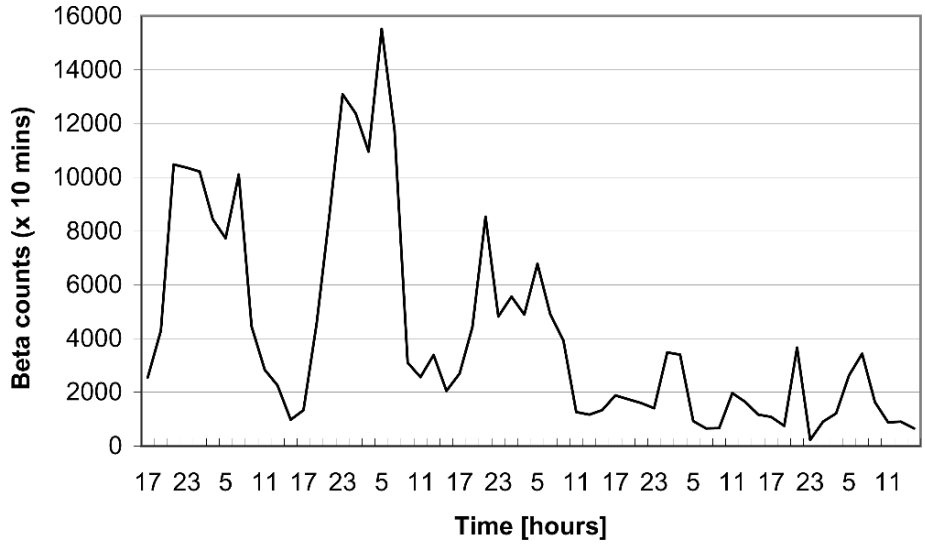


Fig. 13.1 A typical time series of beta counts: the first three maxima refer to stable nights and then one must appreciate a transition to conditions of moderate advection

identified with the height at which a parcel of air coming from the ground halts its free convection, at least in nocturnal stable situations dominated by the thermal factor. They confined their research to some experimental campaigns in a town and used the fundamental phenomenon of the urban heat island that, even in the presence of ground inversions in suburban sites, permits the creation of a shallow mixed layer where convection is not suppressed.

In order to quantify the nocturnal stable layer depth over a town, they used a thermal profile from a radiosonde station in the suburbs and the surface air temperature at the radon detection site inside the town. With a potential temperature method or, equivalently, by means of a simple graphical representation (see Fig. 13.2), they were able to estimate an urban mixing height, h_u, from meteorological data. A statistical analysis show that h_e and h_u are highly correlated, so inducing to think that the box model output represents a correct estimation of the urban mixing height, at least in situations of high nocturnal stability.

In general, we must be aware that h_e and h_u can differ from the real value of the stable layer depth, the equality being valid at the limit of null mechanical turbulence, when the physical features of the BL are completely driven by the vertical thermal state of the atmosphere. Furthermore, while h_u has no chance to be sensitive to nonthermal factors, h_e represents an

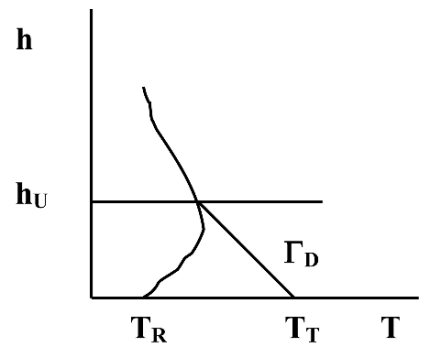


Fig. 13.2 Estimation of the urban mixing height: T_R is the temperature at the radiosonde station, T_T is the temperature inside the town, Γ_D is a dry adiabat drawn from T_T that intercepts the vertical profile

index whose value is determined by all the factors that influence the dilution properties of the BL.

As a final remark, we note that, if direct measurements of Φ are not available, its value can be estimated by inverting equation (13.1) in which we substitute h_e with its meteorological estimation h_u at a certain hour in the evening, at the beginning of accumulation.

13.2.3 An Improved Box Model

Thus, in strong stable situations during the night, we are able to monitor the height of the stable layer

over a town by means of radon detection and the application of a simple box model. This estimation furnishes important physical information about the volume available for the dilution of pollutants emitted at the ground. In particular, this method allowed us to explain critical peak events of primary pollution during the night due to negative fluctuations (reductions) of the stable layer depth. Sometimes, however, the nocturnal fluctuations in h_e, shown by the box model, assume wide unphysical values: then, in order to achieve more realistic modeling of the nocturnal BL behavior and to avoid these problems, a new version of the model has been recently worked out and preliminarily presented in Pasini et al. (2002).

As a matter of fact, the structure of the box model described above is too simple for at least two reasons: first, radon decay is neglected; secondly, entrainment of air with different radon concentrations is not allowed from the top of the box (this is critical just in situations characterized by nocturnal fluctuations of the stable layer depth). A new model structure, which includes these elements, is briefly presented here.

In Fig. 13.3 "compressions" are the situations in which the stable layer depth decreases (i = 1, 2, 3, 6) and "expansions" are the cases when h_e increases (i = 4, 5). In what follows, λ represents the decay constant of radon, Δt is our sampling rate (usually 2 h), C^a is the calculated concentration in the residual layer and we adopt the symbolic form $\Delta h_e(\mathrm{n}, \mathrm{m}) = h_e(\mathrm{n}) - h_e(\mathrm{m})$ for the difference between equivalent mixing heights at time n and m, respectively.

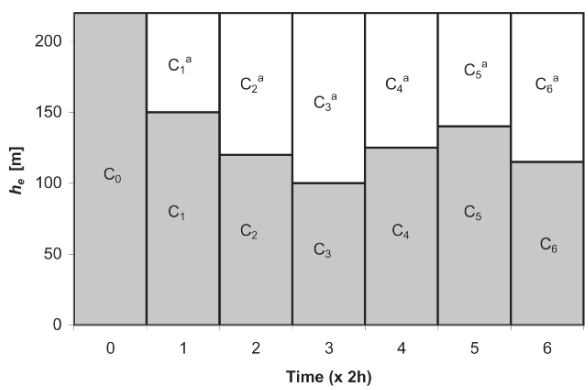


Fig. 13.3 Time evolution of the nocturnal stable layer (gray box). The white area above represents the residual layer

In compression cases the generalization of equation (13.1) reads as follows:

$$h_e(i) = \frac{\Phi}{\lambda} \frac{1 - \exp(-\lambda \Delta t)}{C_i - C_{i-1} \exp(-\lambda \Delta t)} \tag{13.2}$$

The concentration left in the residual layer after the ith compression is:

$$C_i^a = \frac{C_{i-1}^a \exp(-\lambda \Delta t) \cdot \Delta h_e(0, i-1) + C_{i-1} \exp(-\lambda \Delta t) \cdot \Delta h_e(i-1, i)}{\Delta h_e(0, i)} \tag{13.3}$$

If the stable layer depth increases and overlying air is included in the box, i.e. in cases of expansions, the equivalent mixing height can be calculated as:

$$h_e(i) = \frac{(\Phi/\lambda)[1 - \exp(-\lambda \Delta t)] + h_e(i-1)(C_{i-1} - C_{i-1}^a) \exp(-\lambda \Delta t)}{C_i - C_{i-1}^a \exp(-\lambda \Delta t)} \tag{13.4}$$

and the concentration over the top of the box is of course

$$C_i^a = C_{i-1}^a \exp(-\lambda \Delta t) \tag{13.5}$$

This enhanced structure of the box model, tested on past results, leads to a little increase (about 1%) in the value of the linear correlation coefficient between the total set of h_e estimated by the new model and the total set of h_u. The actual, relevant and statistically significant improvement is obtained in critical situations such as nocturnal fluctuations of the stable layer depth, when the application of equations 13.2–13.5 prevents the estimation of wide unphysical oscillations.

Thus, data from radon progeny measurements well represent (in a synthetic way) the dilution properties of the lower layers of the atmosphere and the analysis performed here has led to characterizing these features both qualitatively and quantitatively. In the next section, after having presented another useful application of radon data to primary pollution characterization in an urban environment, we will begin to "handle" these data with a NN model.

13.3 Linear Stability Indices and Their Nonlinear Generalization by NNs

As we have seen, knowledge of radon concentration, or simply of the beta counts coming from the decay of short-lived radon daughters, gives us information about the physical state of the BL. If we consider the so-called primary pollutants, their concentration in the lowest layers of the atmosphere is driven by two main factors: the emission flux and the mixing properties of the BL. In this situation, our knowledge of these physical properties can lead to studying the role of the BL in the accumulation of these pollutants.

13.3.1 Linear Stability Indices

Some years ago, Italian researchers (Perrino et al. 2001) developed stability indices based on radon data in order to characterize days and nights (in every season) in terms of their meteorological predisposition to a primary pollution event.

Substantially, these indices are scalars: they are the results of equations (coming from multiple linear regressions) that aim at reconstructing the mean concentration of benzene on a 12-h interval. The multiple linear regression considered 2-h beta count data and their time derivatives as predictors and benzene concentration during night or day as predictand. For these calculations the year has been divided in three periods, according to different duration and intensity of the solar radiation (winter period: October to February; summer period: May to July; intermediate period: March, April, August, September). A total of six indices has been therefore obtained.

The correlation between the values of these indices and the concentration of benzene is generally good, but it must be highlighted that these indices take into account only one of the two driving forces in determining primary pollutant concentration (the mixing properties of the BL), so that a one-to-one correlation could be possible only in the theoretical case of a constant emission flux. Thus, the scope of indices determination must not be estimating the correct value of observed benzene concentrations, but, more properly, these indices can become a tool for uncoupling the roles of meteorology and emissions for determining the concentration of a primary pollutant.

In fact, these indices allow us to estimate the concentration of a primary pollutant uniquely due to the contribution of the meteorological factor. Therefore, the environmental applications of these indices are especially important in the study of the differences between this estimation and the primary pollutant concentration actually observed. This is of help in identifying days when the atmospheric pollution is heavier (lighter) than predictable on the basis of the atmospheric mixing and allows us to evaluate, for example, the real effect of traffic restriction measures. Also, at a longer range, we are able to understand if a decrease in the air concentration of a given primary pollutant from one year to another is due to a real improvement of the air quality or, instead, only to a lighter meteorological situation.

For further details on the calculation of these linear indices and their application, see Perrino et al. (2001).

13.3.2 Nonlinear Stability Indices via NN Modeling

As stated in the previous subsection, the aim in using the stability indices is not the accurate reconstruction of benzene values in every situation, since these indices take only the BL dilution factor into account and neglect the contribution of emission variations to benzene concentration. With a change of perspective, we recently asked if this meteorological contribution to benzene behavior is correctly modeled (Pasini et al. 2003c). In fact, if we consider that the linear stability indices are obviously not able to fully capture the complex nonlinear relationships among different variables in the BL, we can suggest the use of a more complex nonlinear regression method and hope that its use can lead to an increase of the amount of variance explained by the linear regression. Therefore, after a preliminary statistical analysis of the data available, in order to discover nonlinearities hidden therein, a NN model is applied to the problem of reconstructing benzene concentrations by means of radon data, even in cases not included in the training set.

As far as the neural model is concerned, the NN development environment briefly described at Subsection 12.3.3 of the previous chapter is used even in this case study. We briefly remind that our NNs are multi-layer perceptrons endowed with one hidden

layer and backpropagation training. Furthermore, it is worthwhile to stress that some specific tools are present in our development environment in order to handle historical data and to train NNs starting from quite short time series. We have seen a first example of these training tools in Chapter 12 of this book and we will meet another example in the next section, where a sketch of our normalized sigmoids will be also be presented, together with a discussion of their influence on the model structure.

The application of a nonlinear NN model is suggested by the results of an *a priori* bivariate statistical analysis that is able to estimate linear and nonlinear correlations between each predictor/input (a 2-h radon detection or its time derivatives) and the predictand/target (mean benzene concentration during day or night). By this analysis we compare values of the linear correlation coefficient R and values of the so-called correlation ratio, a nonlinear generalization of R (see Pasini et al. 2001, 2003a for technical details on the fundamentals of such an analysis). Differences are found between linear and nonlinear correlation values for the same input-target sets and, sometimes, inputs showing low linear correlation with the target assume high nonlinear correlation values. In short, even if the correlation ratio does not measure all types of nonlinearity, for our problem it allows us to understand that some nonlinearities are hidden in the relationships among the variables. In particular, as a consequence of this statistical analysis, the inputs considered for an optimal nonlinear regression can be generally different from the variables chosen for an optimal linear regression.

As in the development of linear stability indices, we consider six records of cases (diurnal and nocturnal situations for winter, summer and intermediate periods). Each record is divided into three sets: the first months represent the training set and include the validation set (useful in order to establish the threshold for early stopping and chosen as 15 random days inside these first months), the last month of each record is the test set on which we assess the networks' ability to generalize. When comparing linear and NN models' performance on the test set, we obtain a statistically significant improvement in "simulating" the behavior of day and night mean benzene concentrations: our NN model explains 77% of the variance in benzene data, while the linear model achieves a performance of 68%.

Table 13.1 Performance in the modeling of benzene concentrations on the test sets

Period	Linear model	Neural model
Winter – morning	0.786	0.809 ± 0.038
Winter – evening	0.740	0.812 ± 0.020
Summer – morning	0.558	0.576 ± 0.037
Summer – evening	0.639	0.725 ± 0.031
Intermediate – morning	0.664	0.877 ± 0.012
Intermediate – evening	0.698	0.795 ± 0.038

Source: Adapted from Pasini et al. (2003c). With kind permission of Società Italiana di Fisica.

More specifically, the values of several indices of performance have been calculated and consistent results are obtained on the test sets in the six cases cited above. In Table 13.1 the results of both the linear model and the NN model are shown in terms of the linear correlation coefficient (detected vs. modeled). Here, as in applications described in the previous chapter, the error bars associated with the NN performance come from ensemble runs of the model with different initial random weights, so that each network is able to widely explore the landscape of its cost function, and represent ± 2 standard deviations.

As one can see, the majority of improvements obtained by the application of the fully nonlinear NN model is statistically significant. Furthermore, the calculation of the bias allows us to appreciate that, even in the few cases when the statistical significance of performance improvement is not sure, the systematic error in the results of the NN model is lower than that of the linear model: this gives us more reliable results.

In short, the application of NNs to this problem leads to better modeling the meteorological contribution to the behavior of mean benzene concentration over days and nights in distinct periods of the year, if compared with a multiple linear regression.

13.4 Neural Forecasting of the Radon Concentration and Short-Range Estimation of the Nocturnal Stable Layer Depth

Once the ability of a NN model for characterizing the meteorological-induced behavior of a primary pollutant is shown, it would be interesting to test this model in forecasting BL physical features. In this framework, using the knowledge of radon behavior and applying

the box model previously developed, we concentrate on stable nocturnal situations in the BL.

As a preliminary remark, it is worthwhile to note that the physics of the nocturnal stable layer has been recognized as very complex and modeling in this domain represents a major challenge for physicists of the atmosphere (see, for instance, Mahrt 1998). Thus, once again, an AI empirical data-based method (NN modeling) could help in a framework where dynamical modeling shows drawbacks.

In what follows, several forecasting strategies will be considered and particular attention will be paid to didactical aspects rather than technical details: see Pasini and Ameli (2003), Pasini et al. (2003a, b) for a more detailed treatment.

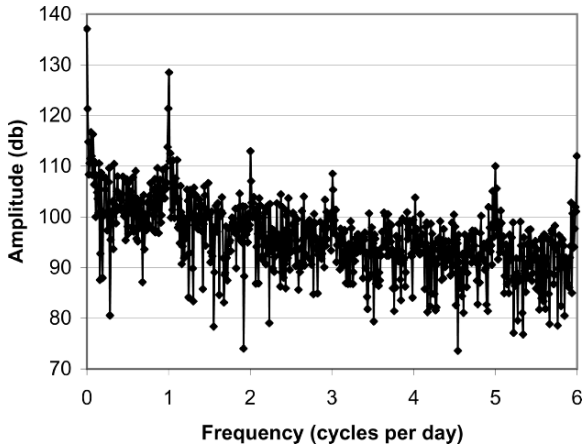


Fig. 13.4 Fourier spectrum of our time series of beta counts

13.4.1 A Time Series Approach: Black Box vs. Preprocessing

The Institute of Atmospheric Pollution of the Italian National Research Council has sponsored several extended duration monitoring campaigns to detect beta counts from radon progeny decay. Thus, long time series at the time resolution of 2 h are available: here, we analyze data from the entire year 1997, detected in a site near Rome, Italy.

Since the development of the model of multi-layer perceptron, NNs have shown their ability to forecast time series data for some steps in the future and they often beat other methods in intercomparison studies: see, for instance, Weigend and Gershenfeld (1993). Therefore, it is natural to apply NNs to a short-range forecast of beta-count time series in order to obtain accurate estimations for the values of this index of the BL dilution capacity.

Usually, when the global dynamics of the system under study is unknown, a NN is used as a black box that outputs future values of the time series, given input of a sequence of its past values. Of course, we can follow this approach even in the treatment of our forecasting problem in the BL: as we will see, this has actually been done with quite good results. Nevertheless, here a part of the dynamics is known: for instance, the influence of a day-night cycle is very clear in situations characterized by nocturnal stability and well developed diurnal mixed layers (look at the first 3 days in Fig. 13.1). Furthermore, a 24-h periodicity and its sub-harmonics are evident in a Fourier spectrum of our time series (see Fig. 13.4). In this situation, this part of dynamics can be modeled or, at least, simply described, leading to using this knowledge before applying a NN for studying the system.

In practice, we chose to preprocess our data. We filtered out the known periodicities (by means of the so-called Seasonal Differencing (SD) method[1]), subtracted this signal from the detected data and modeled the residuals with NNs. In this manner we left the network model the hidden (unknown) dynamics. Of course, the forecasting results of the neural modeling must be added to the preprocessing function in order to obtain final forecasts of the original time series.

The NN tool cited above and briefly described in the previous chapter has also been used in this case study. The choice of inputs is widely discussed in Pasini and Ameli (2003) and the reader can refer to that paper for details. Here we just stress that a particular training-test iterative procedure has been applied. Due to the recognition of a negative effect of old data on forecasting results, we fix a "50-day memory" of training cases and update it for every new forecast, thus limiting the training to the same "season" of any forecast case. Figure 13.5 shows this "moving window" strategy: the window is the 50-day memory (training set), it is updated every 2 h by inserting the latest detected

[1] In our application, SD consists in subtracting from the time series a replica of the series itself delayed by a 24-h time lag. In doing so, we obtain a residual series that is practically lacking in those periodicities shown by the original data.

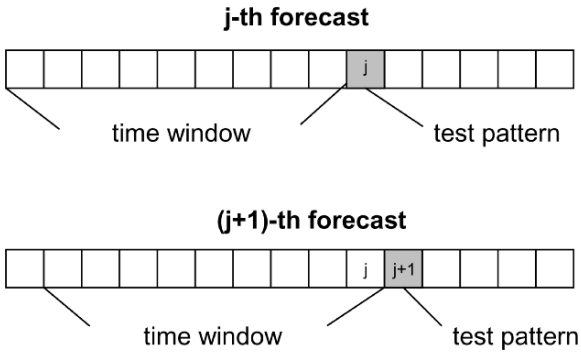


Fig. 13.5 The "moving window" procedure for training and test (forecast) of historical data

data and discarding the oldest ones, then allowing new training and a new forecast.

Once our data is preprocessed by SD, we obtain a residual time series that looks like noise, so that a man-made analysis does not allow us to glean additional clear dynamics in these data. In this situation, the application of a NN model to forecasting this residual time series and to evaluating its forecasting performance could lead to determining whether nonlinear dynamics is hidden in these data, or rather, if they are actually random.

As a matter of fact, we found quite good performance on the test set of residuals when forecasts were extended to 2, 4 and 6 h after the most recent data in the time series. In particular, we calculated the linear correlation coefficient (modeled vs. detected) and obtain $\mathrm{R}(t_0+2) = 0.675 \pm 0.006$, $\mathrm{R}(t_0+4) = 0.500 \pm 0.008$ and $\mathrm{R}(t_0+6) = 0.431 \pm 0.009$. Even if these values are not very high (remember that the time series looks like noise), they still indicate a clear signal of nonlinear dynamics hidden in the residuals. In this way the NNs show their ability in capturing this dynamics and can add information to modeling the system. In particular, they allow us to improve our forecasting performance on the system itself.

Furthermore, a black box strategy and a preprocessing one can be compared in NN modeling on this case study. Table 13.2 shows the performance of the direct application of NNs to forecasting the original time series (TS) and of the joint application of SD and NN forecasting of the residuals (SD-TS): the SD-TS approach outperforms the black box one in all cases and the increment in performance is particularly high at the longer ranges.

Table 13.2 Performance on the test set for the black box (TS) and the preprocessed (SD-TS) strategies in a time series approach

Period	$\mathrm{R}(t_0+2)$	$\mathrm{R}(t_0+4)$	$\mathrm{R}(t_0+6)$
TS	0.812 ± 0.010	0.735 ± 0.017	0.672 ± 0.017
SD-TS	0.894 ± 0.004	0.870 ± 0.004	0.861 ± 0.005

In general, the shape of the signal for beta detections/radon concentrations is well forecasted by SD-TS on all the 24-h interval (nights, days, intermediate periods). In particular, SD-TS prevents the forecasting of counter-tendency behaviors in radon evolution (e.g., increasing beta counts for situations of a detected decrease of beta counts), which are quite often present in the TS forecasts.

If we concentrate on nights with stable conditions in the lower layers, we can apply the box model cited above and obtain forecasts for the depth of the nocturnal stable layer. In Fig. 13.6 an example of h_e 2-h forecasts is presented for three nights: one can recognize the very good results obtained by the joint application of preprocessing and NN model (SD-TS). Due to the very impact of nocturnal BL depth fluctuations on primary pollutants' behavior, a useful feature of this approach is that it generally leads to a correct forecast for the sign of derivatives in h_e records, while the black box approach often presents counter-tendency derivatives in its forecasts.

13.4.2 A Synchronous Pattern Approach with Meteorological Data

In the previous subsection a typical time series approach has been adopted for NN application by using delayed values of beta counts data as input. When the behavior of an index-variable (here beta counts) is influenced by the state of a physical system (here the BL), however, it is a good idea to estimate this state at a time t_0 through the values of certain critical variables and to search for a relationship linking this estimation at time t_0 with future values of the index-variable by means of NNs.

An estimation of the state of the BL can be given through monitoring performed by a standard weather station *in situ*. In our case study, 1-h weather parameters were available from the local meteorological station, so that we considered the following synchronous

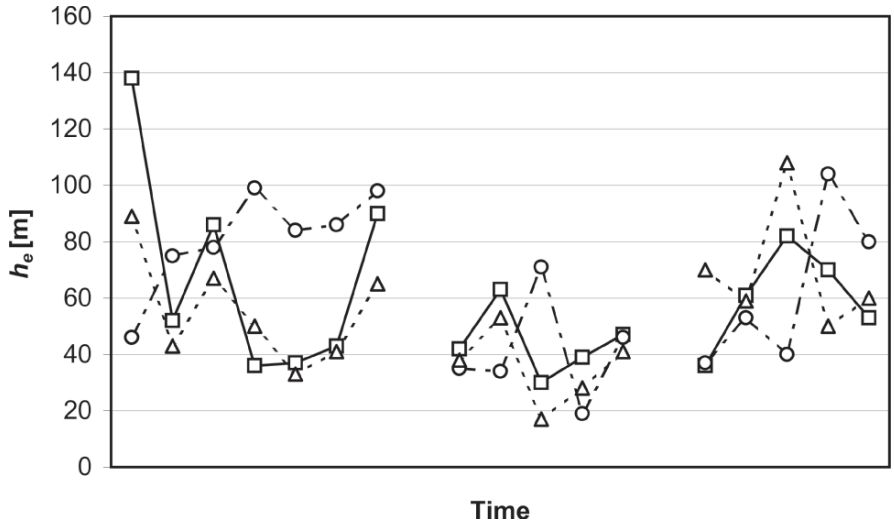


Fig. 13.6 The depth of the nocturnal stable layer calculated by the box model for detected data (square), non-preprocessed forecast values by TS (circle) and preprocessed forecast values by SD-TS (triangles)

pattern of variables which describes the BL state at time t_0: hour of the day (expressed in two inputs as $[\sin(\pi t/12) + 1]/2$ and $[\cos(\pi t/12) + 1]/2$), beta counts, time derivatives of beta counts with respect to 2 h before, sky covering and height of the lowest cloud layer, temperature, dew point, pressure, horizontal wind speed, and visibility.

Thus, the first attempt can be to apply this approach and to analyze the related NN forecasting performance. Linear and nonlinear statistical bivariate analyses between any test set of a single input variable and the set of detected beta counts at $t_0 + 2$, $t_0 + 4$ and $t_0 + 6$ h, however, show that some inputs are poorly correlated with the output. This induces prune some of the inputs.

Pruning is a consolidated technique and the description of its advantages can be easily found in the literature. Here we just stress that, as cited in Subsection 12.3.3 of the previous chapter, the transfer functions in our tool are sigmoids in which the arguments of the exponential function are normalized with respect to the number of connections converging to a single neuron of the hidden and output layer, respectively. For the hidden layer, for instance, we have:

$$g_j\left(h_j^{\mu}\right) = \frac{1}{1 + \exp\left(-\frac{h_j^{\mu}}{\sqrt{n_{hl}}}\right)} \tag{13.6}$$

where n_{hl} is the number of connections converging to a single neuron of the hidden layer.

Figure 13.7 shows the consequences of this normalization on the shape of sigmoids: in practice, it leads to transfer functions which are less nonlinear when one moves from networks with few connections to bigger ones. In short, this leads to different models for different networks, with more nonlinear transfer functions for small NNs: this could even lead to an increase in forecasting performance when pruning is applied in a strongly nonlinear system.

The best performance with a pruned network came from considering: the hour of the day (two inputs), the horizontal wind speed, beta counts, and time derivative of beta counts. Thus two networks with 11 and 5 inputs, respectively, have been applied to this forecasting activity. The performance results are shown in

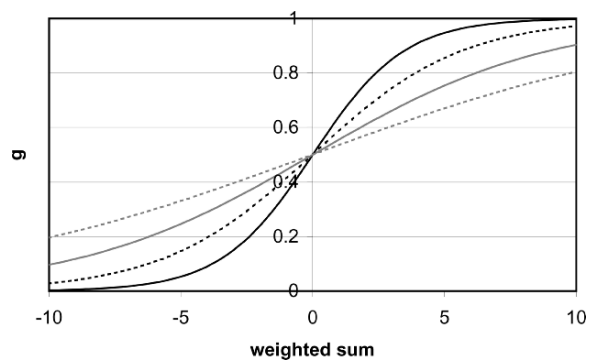


Fig. 13.7 Form of sigmoids in our NN tool for different numbers n of connections converging to a single neuron of the hidden or output layer: solid curve, $n = 3$; heavy dashed curve, $n = 8$; shaded curve, $n = 20$; shaded dashed curve, $n = 50$ Source: Pasini et al. (2001). Copyright AGU

Table 13.3 Performance on the test set in a synchronous pattern approach for the full set of inputs (SP) and in the pruned case (SP-PR)

Period	R($t_0 + 2$)	R($t_0 + 4$)	R($t_0 + 6$)
SP	0.849 ± 0.003	0.821 ± 0.005	0.788 ± 0.006
SP-PR	0.854 ± 0.002	0.820 ± 0.005	0.773 ± 0.007

Table 13.4 Contingency table at a fixed threshold distinguishing between events and nonevents

DET\FOR	No	Yes	Sum
No	a	b	g
Yes	c	d	h
Sum	e	f	n

Table 13.3 in terms of the linear correlation coefficient between modeled (forecasted) and detected beta counts.

A comparison of these results with those presented in Table 13.2 for the time series approach allows us to recognize that the synchronous pattern approach always outperforms the original time series approach (TS) in a statistically significant way. This bears witness to the relevance of meteorological information to better characterize the BL state for short range forecasting. Pruning leads to a little improvement at $t_0 + 2$ h, while the results at $t_0 + 4$ and $t_0 + 6$ h are comparable and worse, respectively, with respect to the runs without pruning.

On the other hand, the recognizing periodicities and preprocessing the time series (SD-TS in Table 13.2) leads to even better performance results, especially at $t_0 + 4$ and $t_0 + 6$ h. This bears witness to the importance of using dynamical information when available.

In Subsection 13.4.4 we will combine the strong points of these two approaches in order to build a hybrid approach that allows us to obtain better forecasting performance. Now, we would like to briefly analyze performance through indices calculated on contingency tables.

13.4.3 Some Measures of Performance

As discussed in Chapter 3 of this book, the problem of performance assessment is manifold. Here we do not enter into details of which index is more appropriate to "measure" the performance in our case. Nevertheless, we show that useful information comes from calculating some indices on contingency tables of events and nonevents related to detected and forecasted beta counts.

As usual in analyzing forecasting performance, we divide detections and forecasts into classes, build contingency tables and assess performance in a dichotomic form. By limiting ourselves to the analysis of the $t + 2$ forecasting performance, we choose 100 equidistant thresholds and divide our range in 100 classes, so obtaining 100 contingency tables. Our notations are referred to Table 13.4, where $a =$ number of nonevents predicted as nonevents, $b =$ number of nonevents predicted as events, $c =$ number of events predicted as nonevents, $d =$ number of events predicted as events. For each threshold we calculated the following indices:

BIAS $= f/h$;
POD (Probability Of Detection) $= d/h$;
FAR (False Alarm Ratio) $= b/f$;
HR (Hit Rate) $= (a + d)/n$;
EFF (EFFiciency) $= (a/g) \times (d/h)$;
CSI (Critical Success Index) $= d/(b + h)$;
HSS (Heidke's Skill Statistics) $= [2(ad - bc)]/(gf + he)$.

Once the values of these indices for each threshold are calculated, one can plot them on graphs built with the value of the threshold as abscissa and the value of the index as ordinate, thus obtaining pictures of global performance for every range of data. This can be done for every forecasting strategy adopted in our case study. Due to the fact that the performance of the synchronous pattern approach with and without pruning are very similar, just SP-PR, TS and SD-TS are considered in these graphs in order to make them more clearly readable. Furthermore, calculations of FAR, HR and CSI reveal that the differences in performance among these three modeling strategies are nearly absent in these indices, so that the related graphs are not shown here. In Fig. 13.8 the results for BIAS, POD, EFF and HSS are shown: they imply some considerations.

The high values of the SD-TS curves for the first low thresholds in BIAS and POD plots account for an overestimation of the low values of beta counts detected. This is primarily due to the transition periods from situations driven by the thermal cycle to

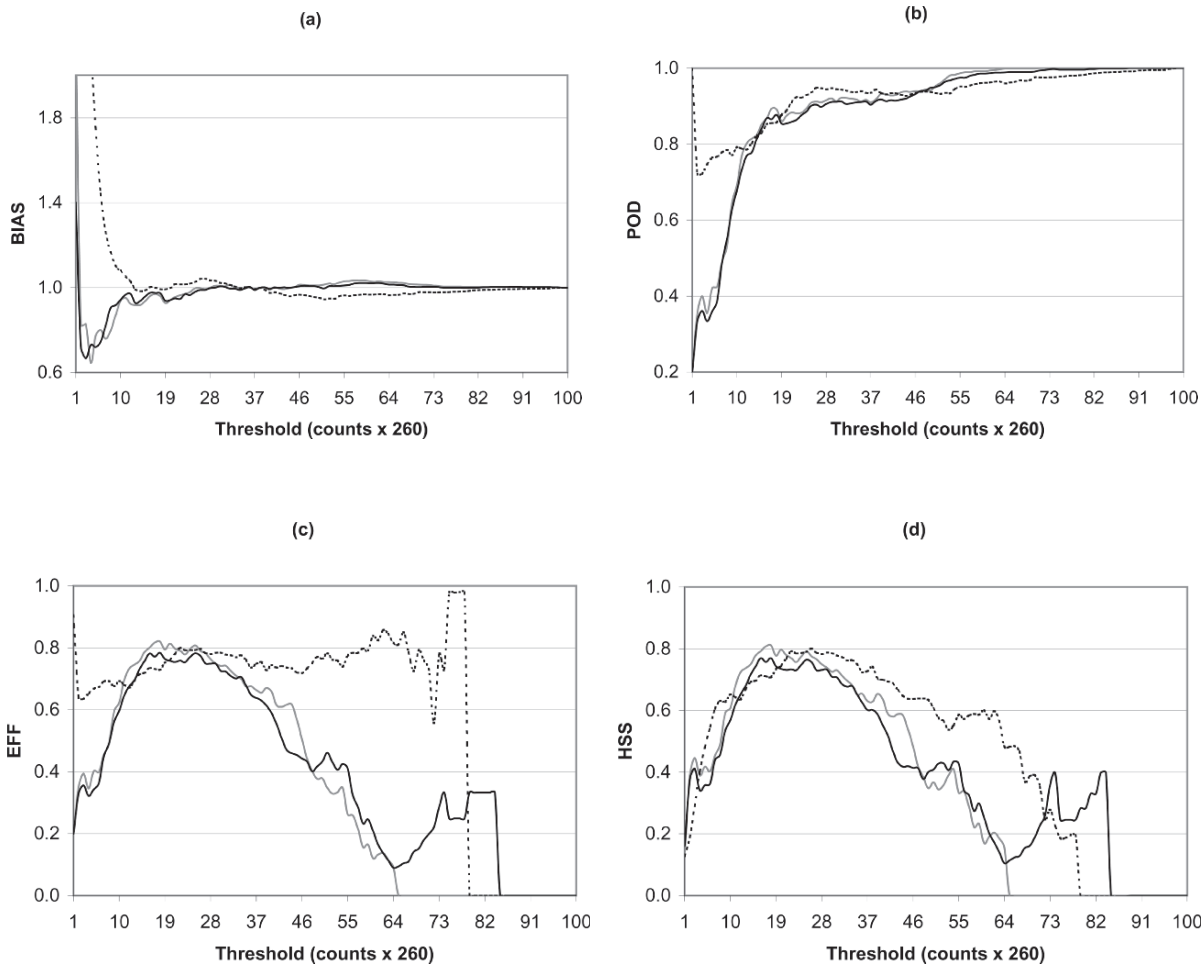


Fig. 13.8 Forecasting performance on the test set of beta counts for the black box time series approach (TS: black curves), the preprocessed time series approach (SD-TS: dashed curves) and the synchronous approach with pruning (SP-PR: gray curves). The four plates refer to BIAS (a), POD (b), EFF (c) and HSS (d)

advective situations when the SD modulation leads to very low final forecast values. Of course, this represents a negative feature of this preprocessing method and suggests adopting some different preprocessing methods during these transition periods.

The graph of EFF (Fig. 13.8c) indicates that the two time series approaches outperform SP-PR at high thresholds. The difference between SD-TS and SP-PR above abscissa 30 (in threshold units) is particularly large.

Undoubtedly, the most interesting results come from analyzing the graph related to HSS (Fig. 13.8d). First of all, HSS is a very good measure of performance (see, for instance, the discussion in Chapter 3 and in Marzban 1998). Furthermore, Fig. 13.8d shows that the time series approach reveals very good features for high thresholds; in particular, SD-TS is always better than other approaches for thresholds >23 abscissa units (about 6,000 beta counts). On the other hand, SP-PR outperforms TS until a threshold of about 46 abscissa units. Finally, we stress that all the approaches lead to a maximum of performance for a threshold around 19 abscissa units (about 5,000 beta counts): this is very important for us, because this value can be considered as the threshold that allows us to distinguish between advective situations and maxima due to the presence of nocturnal stable layers driven by the vertical thermal state of the lower atmosphere.

13.4.4 A Hybrid Approach and Its Forecasting Results

The results discussed above showed that the using a synchronous pattern of meteorological parameters as inputs to the neural model leads to better results than a standard (non-preprocessed) time series approach. This bears witness to the importance of having information about the initial physical state of the BL. On the other hand, using our *a priori* knowledge of day-night periodicities, and the consequent preprocessing activity *via* SD, outperforms both TS and SP strategies.

Taking these results into account, we now wish to explore the possibility that meteorological conditions could give improvements in capturing nonlinearities when periodic contributions to the BL dynamics are subtracted by SD. Therefore, once we recognize that the day-night cycle affects the radon time series as well as time series of meteorological parameters, even if sometimes at a different degree, we apply SD preprocessing to available meteorological data, as well.

Therefore, first we apply SD to the time series of beta counts and to the records of meteorological parameters. Then we concentrate on the following three different choices of inputs for the networks to be trained on the residual series:

- SD-MET: The inputs are the residuals coming from the application of SD to several meteorological variables (wind speed, pressure, temperature, dew point, sky covering and height of the lowest cloud layer, meteorological visibility) and to values of beta counts and the time derivative of beta counts at t_0.
- SD-TS + V: The inputs are the residuals coming from the application of SD to both the delayed values of the time series and the wind speed. Wind is considered very important because it characterizes the mechanical turbulence in the BL and, in particular, the situations of advection, when the periodicity in the time series (essentially due to the day-night cycle) is broken.
- SD-TS + MET: The inputs are the residuals coming from the application of SD on both the delayed values of the time series and the same meteorological variables considered also in SD-MET.

In this subsection our aim is to compare these approaches with the SD-TS approach previously discussed in order to see if the insertion of residuals of meteorological variables into the input layer leads to better results. In doing so, we compare the ability of the distinct networks in capturing the hidden dynamics on just the residual series.

Table 13.5 Forecasting performance on the residual series when SD is applied also to records of meteorological variables

Method	**R($t_0 + 2$ h)**
SD-TS	0.675 ± 0.006
SD-MET	0.571 ± 0.008
SD-TS+V	0.687 ± 0.004
SD-TS+MET	0.679 ± 0.005
Method	**R($t_0 + 4$ h)**
SD-TS	0.500 ± 0.008
SD-MET	0.297 ± 0.006
SD-TS+V	0.528 ± 0.007
SD-TS+MET	0.544 ± 0.006
Method	**R($t_0 + 6$ h)**
SD-TS	0.431 ± 0.009
SD-MET	0.151 ± 0.012
SD-TS+V	0.456 ± 0.008
SD-TS+MET	0.496 ± 0.007

Source: Pasini et al. (2003b). Copyright IEEE.

The results on the test set are presented in a concise form in Table 13.5, where the calculation of the linear correlation coefficient between targets and outputs is reported. Note that, contrary to Tables 13.2 and 13.3, here the forecasting results refer to the NN performance on only the residual series, without any *a posteriori* composition with the SD signals.

A brief analysis of these results indicates that the information on the meteorological parameters alone in the SD-MET approach (even if data on beta counts and its derivatives is included) is not able to capture a satisfying hidden dynamics or to improve forecasting results on the residual series. On the other hand, the same meteorological parameters contribute (in a statistically significant manner) to improving the results of the SD-TS approach when inserted into the input layer together with data on the time series itself. In a certain sense, one could say that the meteorological parameters act as second-order correctors to the forecast obtained by the SD-TS approach. Nevertheless, we must stress again that in a non-linear system we are not able to accurately separate the contributions of each "influence factor" to the final result.

The increase in performance obtained through application of the SD-TS + V and SD-TS + MET approaches is more evident at the longest ranges, so that we can envisage a shift of the predictability horizon for forecasting radon from observations beyond 6

h. Furthermore, we want to stress that networks fed with TS and all the meteorological parameters in the input layer are obviously larger than networks with TS and wind only. The particular structure of our sigmoids allows us to obtain better results in the very short-range with a little network and wind only, which is correlated (linearly and non-linearly) quite well with the target. When, at the following time steps, this correlation decreases, other meteorological inputs become so important to invert the situation of performance scores between SD-TS + V and SD-TS + MET.

13.5 Conclusions and Prospects

In this chapter, the role of NNs has been analyzed for modeling some features of a complex system such as the BL, whose dynamical modeling is very critical (especially in nocturnal stable situations) due to the many interactions and feedbacks that occur therein. In doing so, the BL physical dispersion properties have been summarized by means of a suitable index, the beta counts coming from the decay of short-lived radon progeny. Once this index is identified as a critical variable for describing BL behavior, several approaches have been presented for NN processing its data, with the aim of both BL diagnostic characterization and forecasting.

In this framework, interesting goals have been achieved: we stress the improved forecasting results from jointly applying our knowledge of the day-night cycle's periodicities *via* SD and the NN forecasting of the residual series. Here, in particular, NNs are able to find a hidden dynamics in what appears as a noise signal, leading to a substantial improvement in forecasting performance when compared with a black box NN application. Furthermore, the results of the hybrid approach described in the last subsection appear quite promising.

Once more, the applications described in this paper imply that, at present, the identification of key variables in a complex system and its data-driven modeling by NNs can represent a valid alternative to dynamical modeling.

Of course, although these investigations are preliminary, the scope of this paper was merely to introduce the reader to specific applications of NN modeling in another complex system, the BL, after the previous chapter dedicated to climate. This preliminary research leaves several open questions and directions for further development, considering that, for instance, only very standard NNs have been used and a very simple kind of preprocessing has been adopted. Nevertheless, in an introductory book to techniques and applications, I believe that this is not a fault, but rather a spur for the reader to further research this field.

References

Allegrini, I., Febo, A., Pasini, A., & Schiarini, S. (1994). Monitoring of the nocturnal mixed layer by means of particulate radon progeny measurement. *Journal of Geophysical Research*, *99*(D9), 18765–18777.

Guedalia, D., Ntsila, A., Druihlet, A., & Fontan, J. (1980). Monitoring of the atmospheric stability above an urban and suburban site using sodar and radon measurements. *Journal of Applied Meteorology*, *19*, 839–848.

Lopez, A., Guedalia, D., Servant, J., & Fontan, J. (1974). Advantages of the use of radioactive tracers 222 Rn and 212 Pb for the study of Aitken nuclei within the low troposphere. *Journal of Geophysical Research*, *79*, 1243–1252.

Mahrt, L. (1998). Stratified atmospheric boundary layers and breakdown of models. *Theoretical and Computational Fluid Dynamics*, *11*, 263–279.

Marzban, C. (1998). Scalar measures of performance in rare-event situations. *Weather Forecasting*, 13, 753–763.

Pasini, A., & Ameli, F. (2003). Radon short range forecasting through time series preprocessing and neural network modeling. *Geophysical Research Letters*, *30*(7), 1386. Doi:10.1029/2002GL016726.

Pasini, A., Pelino, V., & Potestà, S. (2001). A neural network model for visibility nowcasting from surface observations: Results and sensitivity to physical input variables. *Journal of Geophysical Research*, *106*(D14), 14951–14959.

Pasini, A., Ameli, F., & Febo, A. (2002). Estimation and short-range forecast of the mixing height by means of box and neural-network models using radon data. In S. Anić (Ed.), *Physical chemistry 2002: Proceedings volumes* (pp. 607–614). Society of Physical Chemists of Serbia: Belgrade, Serbia.

Pasini, A., Ameli, F., & Lorè, M. (2003a). Mixing height short range forecasting through neural network modeling applied to radon and meteorological data. *Proceedings of the 3rd Conference on Artificial Intelligence Applications to Environmental Sciences, 83rd Annual Meeting of the American Meteorological Society*. AMS: Long Beach, CA, CD ROM.

Pasini, A., Ameli, F., & Lorè, M. (2003b). Short range forecast of atmospheric radon concentration and stable layer depth by neural network modelling. *Proceedings of the IEEE International Symposium on Computational Intelligence for Measurement Systems and Applications* (pp. 85–90). Lugano, Switzerland: IEEE.

Pasini, A., Perrino, C., & Žujić, A. (2003c). Non-linear atmospheric stability indices by neural-network modelling. *Nuovo Cimento*, *26C*, 633–638.

Perrino, C., Pietrodangelo, A., & Febo, A. (2001). An atmospheric stability index based on radon progeny measurements for the evaluation of primary urban pollution. *Atmospheric Environment*, *35*, 5235–5244.

Porstendörfer, J. (1994). Properties and behaviour of radon and thoron and their decay products in the air. *Journal of Aerosol Science*, *25*, 219–263.

Seinfeld, J. H., & Pandis, S. N. (1998). *Atmospheric chemistry and physics: From air pollution to climate change* (1326 pp.). New York: Wiley.

Stull, R. B. (1988). *An introduction to boundary layer meteorology* (666 pp.). Dordrecht: Kluwer.

Vinod Kumar, A., Sitaraman, V., Oza, R. B., & Krishnamoorthy, T. (1999). Application of a numerical model for the planetary boundary layer to the vertical distribution of radon and its daughter products. *Atmospheric Environment*, *33*, 4717–4726.

Weigend, A. S., & Gershenfeld, N. A. (Eds.) (1993). *Time series prediction: Forecasting the future and understanding the past* (664 pp.). New York: Addison-Wesley.

Addressing Air Quality Problems with Genetic Algorithms: A Detailed Analysis of Source Characterization

14

Sue Ellen Haupt, Christopher T. Allen, and George S. Young

14.1 Introduction

14.1.1 Fitting the Model to the Purpose

The purposes for modeling air contaminants have evolved, and the models themselves have co-evolved to meet the changing needs of society. The original need for air contaminant models was to track the path of pollutants emitted from known sources. Therefore, the initial purpose of the models was to track and estimate the downwind transport and dispersion (T&D). Since dispersion results from turbulent diffusion, which is best modeled as a stochastic process, most models for the dispersion portion are based on a Gaussian spread.

Because many environmental problems have their sources in a region that is far from the impact, there came a need to identify remote sources of pollution. For instance, the acid rain problem that was highly studied in the 1980s was widely thought to be caused by upwind polluters. Power plant emissions in the Ohio Valley were blamed for acid rain in New York and New England. To test this conjecture, receptor models were developed. This type of model begins with monitored pollution concentrations and back calculates the sources. Some models of this type were based on a backward trajectory analysis while others separated out the mix of chemical species present in the sample and computed likely sources given knowledge of the species composition of the potential sources. These models pointed to the Ohio Valley for the source of the acid rain precursors. Receptor models are still popular for attributing pollutants to their sources.

A more recent application analyzes the impact that a toxic release of chemical, biological, radiological, or nuclear (CBRN) material might have on a nearby population. Such a release could be due to an accident at a nearby plant or in transit, intentional release by a terrorist, or enemy action in a military situation. In such cases, there is often a need for a full spectrum of modeling, beginning with characterizing the likely source, then estimating contaminant transport and dispersion as well as their uncertainty, and finally estimating the impact on nearby populations and facilities for the purpose of deciding how best to respond to the situation. Such scenarios require rapid models for source characterization, T&D, and human effects (mortality rates, casualty rates, etc.); so including the full physics and dynamics is computationally prohibitive. Therefore, faster artificial intelligence techniques may become competitive. But such techniques are only as good as the dynamics-based models on which they are built. We show here how a genetic algorithm has proven useful for coupling backward looking receptor models

Sue Ellen Haupt (✉)
Applied Research Laboratory and Department of Meteorology, The Pennsylvania State University, P.O. Box 30, State College, PA 16802, USA
Phone: 814/863-7135; fax: 814/865-3287;
email: haupts2@asme.org

Christopher T. Allen
Computer Sciences Corporation, 79 T.W. Alexander Drive, Building 4201 Suite 260,
Research Triangle Park, NC 27709, USA
email: callen24@yahoo.com

George S. Young
Department of Meteorology, 620 Walker Building,
University Park, PA 16802, USA
Phone: (814) 863-4228; fax: 814/865-3663;
email: young@meteo.psu.edu

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

with the forward T&D models to leverage the strengths of each in addressing the source characterization problem. We demonstrate here how to characterize the strength, location, height, time, and meteorological conditions of a release given field data. We begin by demonstrating the GA-based technique using synthetic data and a very basic T&D model and progress toward incorporating a realistic advanced applications T&D model and validating the techniques with field experiment data.

14.1.2 The Problem of Turbulent Dispersion and Real Data

Pollutant released into a turbulent atmospheric boundary layer is subject to chaotic motions on a variety of scales in both time and space. Thus, we cannot definitively predict an exact concentration for a specific location at an instant in time. As a result, predictive T&D models typically compute an ensemble average by solving a diffusion equation to yield a Gaussian spread. We must remember what such a model can and cannot do. It can predict an expected ensemble mean concentration and its standard deviation. It cannot, however, predict the expected concentration for a specific realization (Wyngaard 1992).

In contrast, concentration measurements represent a specific realization of turbulent dispersion. Currently, there is not a good evaluation method for comparing the single realization of a field experiment with the ensemble average statistics from model output (NRC 2003).

In addition to this stochastic variability of time averages, the pollutant emission rates are often poorly characterized; therefore, the dispersion problem appears intractable. Here we detail a method that uses artificial intelligence to directly treat the problem of inherent uncertainty through coupling a dispersion model to a receptor model. The goal is to blend the predictions of the T&D models with the monitored data, which are grounded in reality. Since blending these two disparate models becomes a complex optimization problem, the genetic algorithm (GA) is an appropriate tool to couple the field measurements to the dynamically based T&D model.

The GA-coupled model described here has evolved in parallel with the focus of the application. The initial formulation was for characterizing sources of air pollution by using the GA to link a forward T&D model with a backward looking receptor model. That model is described in detail in Section 14.2. Two applications with synthetic data are presented in Section 14.3: the first is in an artificial simple geometry and a second is in a realistic geometry. Although the model is shown to perform well, we immediately notice some cases for which the model is ill-posed. A statistical analysis of model performance appears in Section 14.4, which also describes model performance in the presence of random noise. In these initial sections, our goal is to apportion the fraction of monitored pollutant to each of a list of pre-identified sources. In Section 14.5, we begin to address the issues that are relevant for homeland security: what if we don't have a list of candidate sources and what if the local meteorological conditions aren't known? In this case, we apply the GA directly to identify the location, strength, and time of the release as well as to determine the direction of the wind that is transporting the contaminant. To accomplish this feat requires multiple receptors, each monitoring concentrations as a function of time. Section 14.6 is devoted to making the GA coupled model more realistic by incorporating a highly refined T&D model, SCIPUFF. This refinement requires reformulating the model to minimize calls to SCIPUFF and to optimize GA performance. With this refinement, we are able to examine model performance on actual field-monitored data, also presented in Section 14.6.

This problem of source characterization and characterizing the meteorological conditions is a very practical one that several government agencies are addressing. The application of the genetic algorithm to this problem demonstrates the real world applicability of artificial intelligence to such problems.

14.2 Coupled Model Formulation

The purpose of the coupled model is to assimilate field monitored data and back calculate the source characteristics of the emission. Several previous investigators used information on dispersion or chemical transport in computing source apportionment. Qin and Oduyemi (2003) apportioned particulate matter to its sources by using a receptor model and incorporating

dispersion model predictions from vehicle emission sources. Cartwright and Harris (1993) used a GA to apportion sources to pollutant data monitored at receptors. Loughlin et al. (2000) also used a GA to couple an air quality model with a receptor model. They minimized the total cost of controlling emission rates at over 1,000 sources in order to design cost effective control strategies to meet attainment of the ozone standard. Kumer et al. (2004) estimated apportionment factors that match monitored data by combining factor analysis-multiple regression with dispersion modeling. We describe here how we have built on these prior studies to couple a Gaussian plume model with a receptor model via a genetic algorithm to compute the source calibration factors necessary to best match the measured pollutant (Haupt and Haupt 2004; Haupt 2005; Haupt et al. 2006; Allen et al. 2006, 2007). Camelli and Lohner (2004) computed the location of a source that would cause the maximum amount of damage using a GA and a computational fluid dynamics model. Note that all the works mentioned here use Artificial Intelligence (AI) techniques to solve a difficult problem of blending two types of models.

14.2.1 Model Formulation

One method to apportion monitored concentrations to the expected sources is with a chemical mass balance (CMB) receptor model. Such a model starts with receptor data consisting of different monitored chemical species and a list of the emission fractions for each of those species for the potential sources in the locale. Mathematically:

$$\mathbf{C}_{mnr} \cdot \mathbf{S}_n = \mathbf{R}_{mr} \tag{14.1}$$

where C_{mnr} is the source concentration profile matrix denoting the fractional emission from source n; R_{mr} is the concentration of each species measured at a receptor r, and S_n is the apportionment vector, also called calibration factors, to be computed. Subscript n denotes the source number, m the species index, and r the receptor number. The monitored data provides the R_{mr} matrix denoting the amount of each chemical species present at receptor r. If the chemical composition of the emissions from each source is known, a fit to the data produces the fractional contribution from each source, S_n. Although our coupled model is inspired by the CMB model, we do not assume mass fractions of different species, but rather substitute varying meteorological periods. That is, m denotes the meteorological period for our reconfigured model. In the coupled model framework, the emission fractions in C_{mnr} are replaced with pollutant concentrations predicted by a T&D model at each receptor for each meteorological period. The receptor data R_{mr} matches the same meteorological periods. The vector, S_n, apportions or calibrates the expected transport model dispersed emissions to match actual concentrations measured at the receptor.

14.2.2 The Solution Method – A Continuous Genetic Algorithm

While one might begin to solve (14.1) with standard matrix inversion methods, one would quickly discover that the matrix is usually poorly conditioned and not easily inverted. This poor conditioning results because the meteorological periods are seldom independent. Therefore, we pose it as an optimization problem. Traditional optimization methods such as least squares and conjugate gradient perform poorly (Haupt et al. 2006). We did find, however, that other iterative methods such as the Moore-Penrose pseudoinverse (Penrose 1955) can produce an accurate solution for some of the simpler problems solved here. When we tried that method with more complex configurations, it did not produce a viable solution (Haupt 2005). In addition, we aim toward optimizing more than just the source calibration factors (see Section 14.5), so we expect the optimization problem to progress well beyond a matrix solution. Thus, we require a very robust optimization method that can solve this difficult matrix problem while simultaneously estimating other unknown parameters. We achieve this by using a GA as the coupling mechanism that minimizes the difference between the monitored concentrations and the predicted concentrations. The GA was introduced in Chapter 5 of Part I. The continuous GA is appropriate for application to this problem since all parameters are real continuous numbers.

The cost function used here measures the root mean square difference between the left-hand side of (14.1) and the right-hand side, summed over the total number

of meteorological periods considered and normalized. This normalized residual is:

$$\text{Cost} = \frac{\sqrt{\sum_{m=1}^{M}\sum_{r=1}^{R}(C_{mnr}\cdot S_n - R_{mr})^2}}{\sqrt{\sum_{m=1}^{M}\sum_{r=1}^{R}R_{mr}^2}} \tag{14.2}$$

where M is the total number of meteorological periods and R is the total number of receptors. We assume that R_{mr} are monitored data. Thurs, the crux of the model is now to use an appropriate transport and dispersion model to estimate C_{mnr}.

14.3 A First Validation

There are many ways to estimate the dispersed emissions that form C_{mnr}. The first validation problem uses Gaussian plume dispersion:

$$C_{mn} = \frac{Q_{mn}}{u\sigma_z\sigma_y 2\pi}\exp\left(\frac{-y_{mn}^2}{2\sigma_y^2}\right)\left\{\exp\left[\frac{-(z_r-H_e)^2}{2\sigma_z^2}\right] + \exp\left[\frac{-(z_r+H_e)^2}{2\sigma_z^2}\right]\right\} \tag{14.3}$$

where: C_{mn} = concentration of emissions from source n over time period m at a receptor location

(x, y, z_r) = Cartesian coordinates of the receptor in the downwind direction from the source

Q_{mn} = emission rate from source n over time period m

u = wind speed for meteorological period m

H_e = effective height of the plume centerline above ground

σ_y, σ_z = dispersion coefficients in the y and z directions, respectively

The dispersion coefficients are computed from Beychok (1994).

$$\sigma = \exp\left\{I + J\left[\ln(x) + K(\ln(x)\right]^2\right\} \tag{14.4}$$

where x is the downwind distance (in km) and I, J, and K are empirical coefficients dependent on the Pasquill Stability Class, in turn dependent on wind speed, direction, and solar radiation. The coefficients can then be looked up in tables (Beychok 1994).

Initially, we consider data from a single receptor but allow for multiple potential sources of the pollutant to be apportioned. Thus, the receptor index, r, in (14.3) collapses to 1.0 and no longer needs included for this first problem. The pollutant predicted by the forward model form C_{mn} of (14.3) and the monitored data become the right hand side, R_m, that is, the monitored data for the same meteorological periods. The meteorological periods are common to those metrics. The remaining vector, S_n, is thus the source calibration factor, which is tuned to optimize agreement between the model predicted concentrations and the receptor observations. If we had perfect world knowledge (of source characteristics, dispersion processes, meteorological conditions, turbulence, and monitored concentrations), S_n would be composed of all 1.0s. Therefore, the difference of this factor from 1.0 can be interpreted as an error or uncertainty in the modeling process in comparison to the monitored data.

Figure 14.1 summarizes the coupled model process. Given assumed geometry, meteorology, and emissions concentrations, we compute each source's dispersion plume. Then we estimate the contribution of each plume to the total concentration at the monitor with the forward model, which fill the concentration matrix. The monitor has recorded actual concentrations, R_m, for the same meteorological periods. The computed calibration factors then assign the portion to each source. Total emissions from a source can then be computed by multiplying the originally assumed emission rate by the source calibration factor.

14.3.1 Synthetic Data on a Circle

The coupled receptor/dispersion model technique was first validated in a simple geometry. A receptor was sited at the origin of a circle and 16 sources were spaced every 22.5° at a distance of 500 m. Receptor data were generated using the same dispersion model to be used for the coupled model optimization (equations (14.3) and (14.4)). This approach is sometimes called an identical twin experiment (Daley 1991). To estimate the calibration factors for 16 sources requires at least 16 independent meteorological periods. This independence was achieved by using wind directions from 16 points of the wind rose and representative

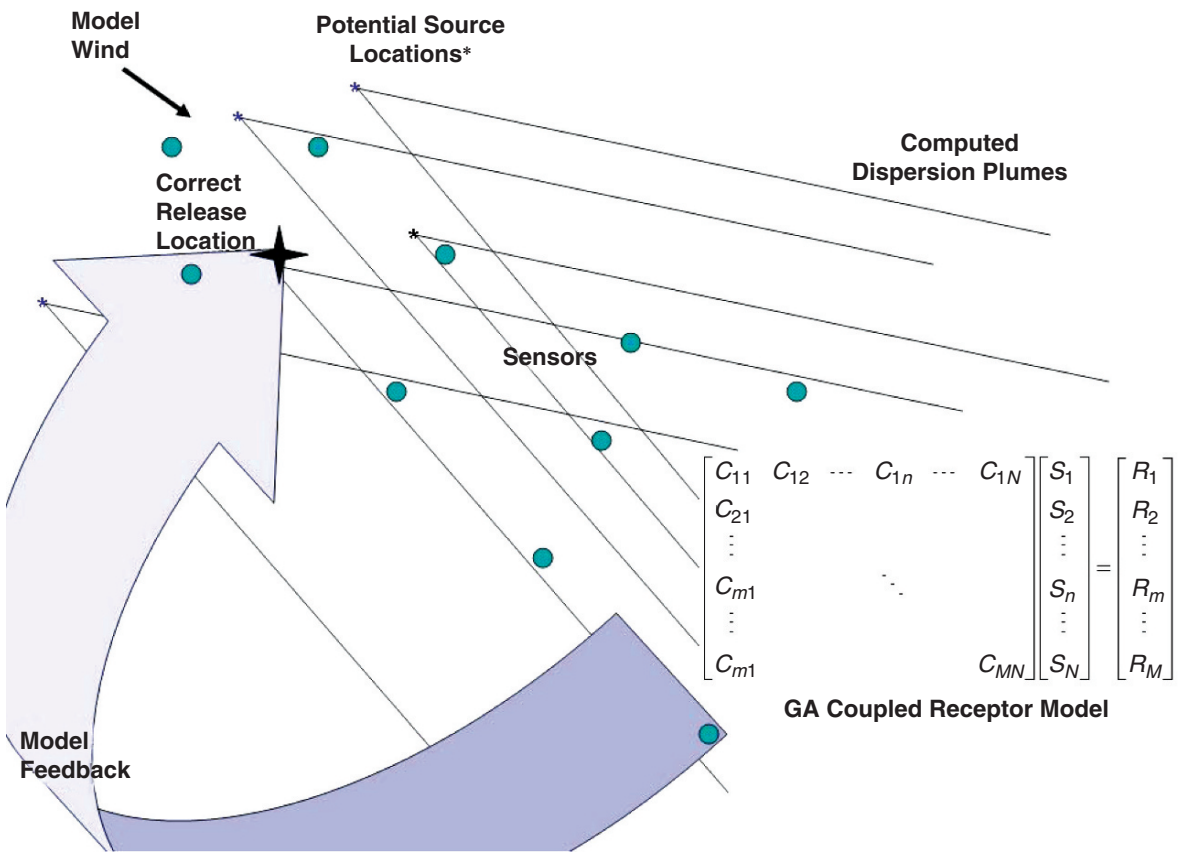


Fig. 14.1 Schematic of the GA coupled model. The monitored receptor data appears in matrix R, the concentration estimates in matrix C, and the GA computes the apportionment vector S to identify sources

wind speeds. Neutral stability was assumed for ease of comparison. The dispersion model was run using 1 h averaging over the meteorological data and specifying calibration factors, S_n, that we hoped to match with the coupled model.

The coupled receptor/dispersion model was then tested with this synthetically generated data. The first tests set calibration factors to 0.0 except for a single source that was set to a 1.0 to simulate identifying which single source might cause a contaminant event. The genetic algorithm, when run with a sufficient number of iterations, successfully evolved the correct solution. For this problem, the number of iterations determines the smallness of the cost function. Figure 14.2 shows the GA convergence over 200,000 iterations, much more than would be used in a typical run. The decrease in the residual is monotonic. So how many iterations are actually necessary or useful to get an acceptably small residual with a reasonable amount of computer time? Table 14.1 shows the results of a sensitivity study of residuals versus the number of iterations. Since the GA randomly generates the initial set of solutions, a different residual is expected for each run. Thus, for this table, each configuration was run five times and the mean and standard deviation of the residuals is listed. The results of Fig. 14.2 are confirmed: more iterations result in a smaller mean residual. The standard deviation also decreases with the number of iterations. Thus we expect this method to produce reliable results with a moderate number of iterations. If we are able to average multiple runs or to use a large number of iterations, the GA is even more likely to converge to a reliable solution. We now have confidence in our approach. Similar results hold for other configurations with two or more sources contributing.

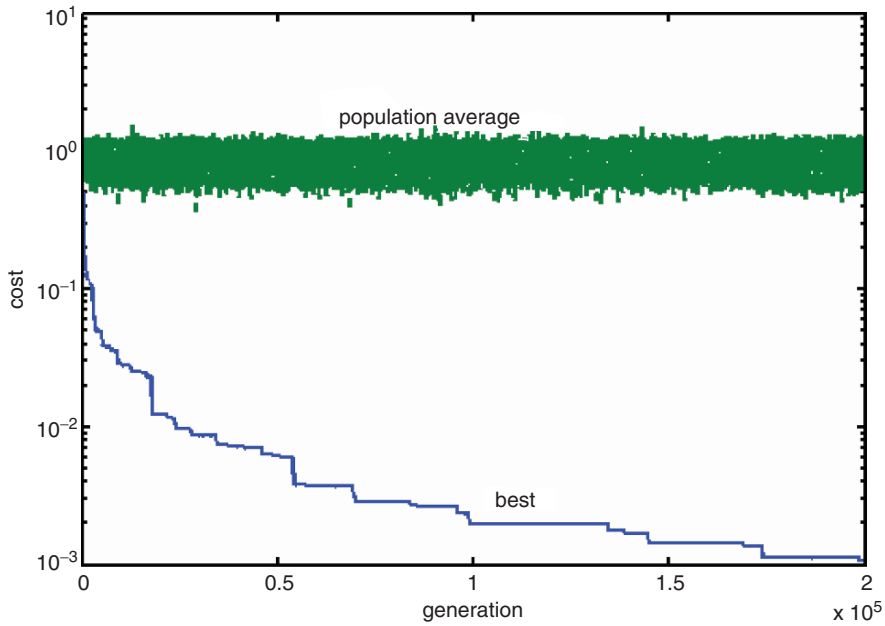


Fig. 14.2 Convergence for GA solution to the circular geometry problem

14.3.2 Actual Emission Configuration with Synthetic Meteorological Data

A second identical twin experiment used an actual emission configuration for Cache Valley, Utah. The source locations were obtained from the state of Utah emission inventory and source heights were estimated. Each source was assigned the same artificial emission rate. The receptor location is the actual monitor located on Main Street in Logan, Utah. Table 14.2 details the source data relative to the monitor. For verification purposes, the meteorological data were produced synthetically to systematically sample the range of possible winds. Source apportionment factors were assumed, once again assigning a factor of 0.0 to all except those chosen for a synthetic emission.

Table 14.1 Residual size as a function of the number of GA iterations for the circular source configuration. Statistics for all but the last row are based on five separate runs

Iterations	Best residual	Mean residual	Standard deviation
500	0.179	0.269	0.096
1,000	0.155	0.191	0.030
2,000	0.052	0.077	0.034
5,000	0.034	0.052	0.020
50,000[1]	5.36×10^{-4}		

[1]Based on a single run.

Using a real source configuration is a much more difficult problem than placing sources in a concentric circle. For instance, consider the case where two sources lie at the same angle from the receptor but at different distances. If the wind speed was not variable, it would be impossible to distinguish between the contributions from those two sources and so the problem would be ill-posed. Thus, we use a variety of meteorological conditions to produce a correct allocation of source apportionment factors.

Table 14.2 Source configuration for Cache Valley, UT

Source number	Distance from monitor (m)	Angle from monitor (° from north)
1	1,492	26.4
2	25,031	8
3	8,550	176
4	25,096	8
5	25,700	9
6	4,789	350
7	13,854	5
8	6,030	178
9	2,227	171
10	9,998	4
11	11,540	245
12	23,285	5
13	2,328	55
14	13,802	4
15	17,994	152
16	569	71

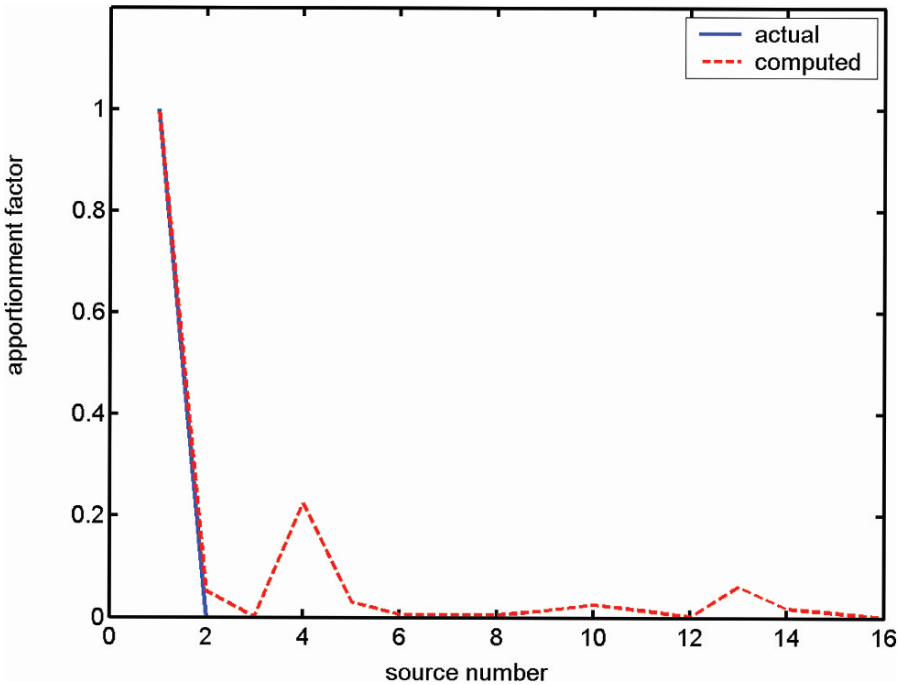


Fig. 14.3 Source apportionment for a single source (source 1) for the Cache Valley, UT configuration

The first validation example had only one source factor of 1.0 so only a single source contributed to the observed concentration. The source chosen was 1.5 km west of the receptor, a direction with no other sources. As seen in Fig. 14.3, the algorithm identified the correct source in less than 10,000 iterations. Some spurious contribution was attributed to source 4; but since that source is 25 km away, its contribution would be well dispersed by the time it reached the receptor. The normalized residual for this run was quite small: 0.0047604.

A second example was intentionally made more difficult to solve by setting an apportionment factor of 1.0 for three sources while the rest were assigned 0.0. The apportionment factors were optimized by the coupled model using 64 meteorological periods and 10,000 iterations. The results appear in Fig. 14.4. The three sources that were given 1.0s were well captured. An additional four sources were spuriously assigned large apportionment factors, in spite of the relatively small residual of 0.070144. Three of those, sources 2, 4, and 5, are located 23–25 km away from the receptor. Thus, their contribution was likely to disperse to a nearly zero concentration by the time it reached the receptor using the Gaussian plume model. Their apportionment factors, when multiplied by near zero, have little impact on the residual and are meaningless. Source 12 is 8.5 km away and therefore more likely to contribute, but it is in the same direction as the three sources that are making a real contribution. The lack of directional distinction makes it difficult to correctly identify only those sources that contribute to receptor pollutant concentration with the current configuration of the coupled model. The problem is depicted in Fig. 14.5. If a source that is 2 km from the receptor is much stronger than one that is only 1 km from the receptor, either could produce an equivalent concentration.

14.3.3 Model Sensitivity to Cost Function Formulation

Would a different formulation of the cost function produce different results? A cost function with a higher power on the difference than the root mean square (RMS) value in (14.2) would weight the outliers more heavily. To evaluate how this might impact the results, we look at alternate formulations for the cost function.

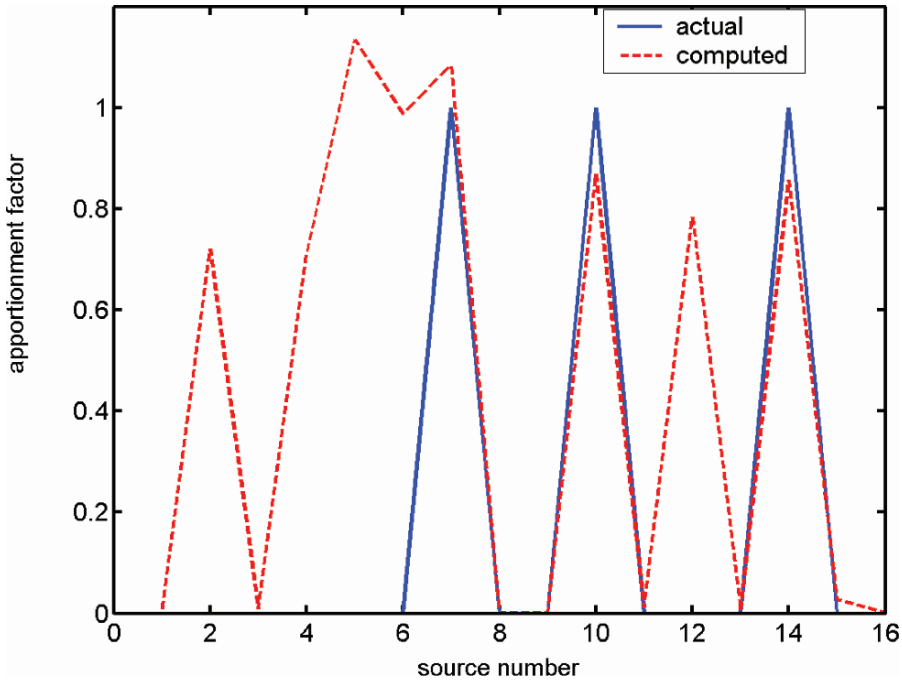


Fig. 14.4 Source apportionment for three sources (sources 7, 10, and 14) for the Logan, UT configuration

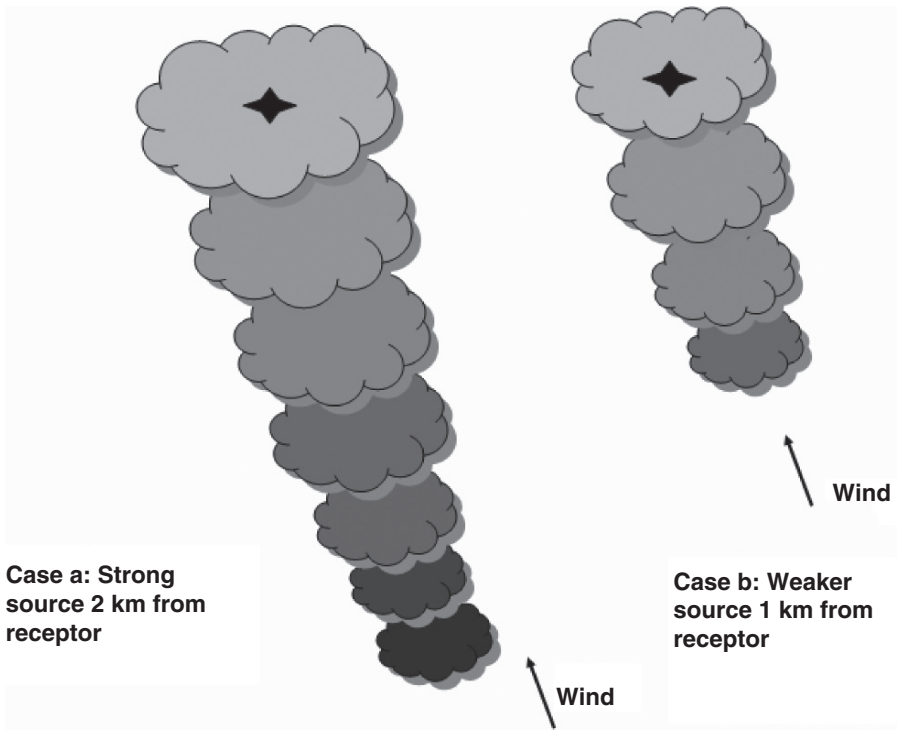


Fig. 14.5 Schematic of plume from two different sources at the same wind angle

Table 14.3 Evaluation of different cost function formulations for a circular geometry

Case & Metric	RMS	SqRoot	AbsVal	FourthRoot	EighthRoot	RMSAbs
RMS	0.050919	0.048137	0.044658	0.056269	0.063798	0.049764
Max	1.02305	1.02045	1.02457	1.02503	1.0352	1.02443
Min	0.97063	0.9727	0.97757	0.97215	0.97195	0.97546
In 0.01	10.5	11.2	11.3	8.0	9.8	11.2

The normalization method makes no difference since the GA mating function used here is based on ranking rather than absolute difference. The formulation of the cost function's numerator, however, could make a difference in the results or in the convergence properties of the model. We showed in Fig. 14.2 that, for this problem, the more GA iterations performed, the lower the cost function. We choose to lump accuracy and convergence properties into a single issue by holding the number of iterations in each GA coupled model run to 20,000.

Five additional cost function formulations are considered:

$$\text{SqRoot} = \frac{\left(\sum_{m=1}^{M}\sqrt{|C\cdot S-R|}\right)^2}{\left(\sum_{m=1}^{M}\sqrt{|R|}\right)^2} \tag{14.5}$$

$$\text{AbsVal} = \frac{\sum_{m=1}^{M}|C\cdot S-R|}{\sum_{m=1}^{M}|R|} \tag{14.6}$$

$$\text{FourthRoot} = \frac{\sqrt[4]{\sum_{m=1}^{M}(C\cdot S-R)^4}}{\sqrt[4]{\sum_{m=1}^{M}(R)^4}} \tag{14.7}$$

$$\text{EighthRoot} = \frac{\sqrt[8]{\sum_{m=1}^{M}(C\cdot S-R)^8}}{\sqrt[8]{\sum_{m=1}^{M}(R)^8}} \tag{14.8}$$

$$\text{RMSAbs} = \text{RMS} + \text{AbsVal} \tag{14.9}$$

Table 14.3 summarizes the results for the circular geometry with all sources assigned a calibration factor of 1.0. The results reported there are for the average of six coupled model runs of 20,000 iterations each. The four different metrics used are:

1. RMS: The RMS difference from the calibration factor that was used to create the synthetic data. We hope to see this minimized.
2. Max: The maximum calibration factor for each run, averaged over the six runs. We hope to see this as close to the actual as possible (1.0 for the circle).
3. Min: The minimum calibration factor each run, averaged over the six runs. We again hope to see this as close to the actual (1.0) as possible.
4. In 0.01: The number of sources calibrated within 1% of actual. A higher value for this metric implies a better result. For the circle case that includes 0.0 apportionment factors, this means within 1% of 1.0.

As seen in the table, there is no clear winner among the cost functions, although the higher power cost functions perform somewhat worse than the SqRoot, AbsVal, RMS, and RMSAbs. For the circular configuration, the AbsVal function works best, closely followed by the SqRoot. For a different geometry the results were somewhat different, but performance differences between the cost functions are relatively small.

A few runs of the GA coupled model with 200,000 iterations for the RMS and AbsVal cost functions confirmed the results of Table 14.3. Thus, although genetic algorithm results can be sensitive to formulation of the cost function, for this problem, any of the cost functions described above will give similar results. We conclude that our original choice of an RMS cost function was reasonable and easy to compare with other methods that are based on RMS differences (Haupt et al. 2006).

14.3.4 Tuning the GA to the Problem

We saw that for both a simple geometry and for a more realistic geometry, the coupled model is able to correctly apportion concentrations to sources in spite of a few spurious apportionments for the most difficult

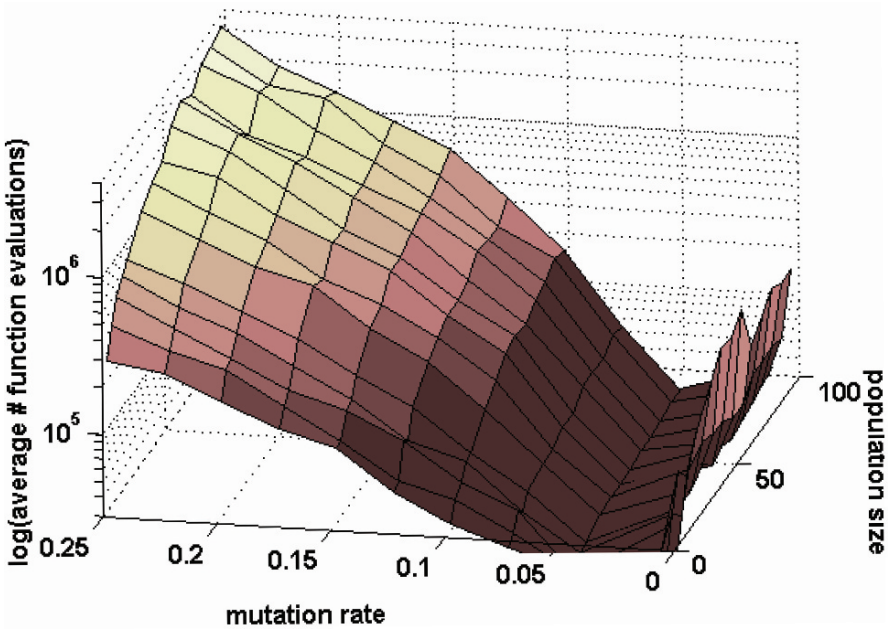


Fig. 14.6 The number of cost function evaluations required to reach a tolerance of 0.01 as a function of population size and mutation rate, averaged over 10 runs

situations. Now we wish to analyze which combinations of GA parameters optimize model performance. When we move to more refined dispersion models (Section 14.6 below), we expect the computation of the dispersion matrix to be computationally expensive, so we wish to determine the best combinations of population size and mutation rate to minimize the number of calls to the cost function, similar to the analysis given in Chapter 5. We noted that for this problem, the GA convergence depends on the number of iterations. Since the solution is known for these identical twin experiments, we can stop the GA when the error has reached a pre-specified tolerance level, in this case 0.01. We wish to explore a wide range of parameter combinations. The goal is to minimize the number of cost function evaluations required to reach this level in an effort to minimize the CPU time. Mutation rates examined are 0.001, 0.005, 0.01, 0.05, 0.075, 0.1, 0.125, 0.15, 0.175, 0.2, 0., and 0.25. Population sizes are 4, 8, 12, 16, 20, 32, 40, 48, 56, 64, 72, 80, 88, and 96. We run the GA for each combination of population size and mutation rate and count the total number of calls to the cost function to achieve convergence (population size times the number of generations, reduced by the number of members that have not changed from one generation to the next). Since the convergence of the GA progresses differently with each random initialization, we average ten separate runs for each mutation rate/population size combination to produce the results in Fig. 14.6. It shows that for this problem, there are various ways to combine population size with mutation rate to produce fast convergence. One way is to use relatively small mutation rates (order of 0.01). The other is to use moderately small population sizes (8–20), even with larger mutation rates (0.15 to 0.2) such as we did in the previous runs. The lowest average number of function evaluations occurred when the mutation rate was 0.05 and the population size was 12. Such a configuration for running the GA is sometimes referred to as a micro-GA due the small population size. These results are similar to those of Chapter 5. Parameter ranges such as these tend to emphasize the impact of mutation and are preferred when there are multiple closely spaced local minima. When such parameter combinations are used, the emphasis is on finding the single best solution rather than evolving the entire population. This is why the mean residual in Fig. 14.2 remained relatively constant in spite of the rapid decrease for the best solution. Note that using elitism, which maintains the best individual

in the population unchanged, is essential for such applications.

The analysis presented here has assumed a serial computer. The analysis would be quite different if a large number of parallel processors were available and the GA was coded to take advantage of them as discussed in Chapter 5.

14.4 Statistical Analysis of Model Performance

14.4.1 The Monte Carlo Approach

This statistical analysis revisits the circular geometry consisting of a single receptor surrounded by 16 potential sources at a radius of 500 m. As discussed above, a single run of a GA coupled model is typically sufficient to estimate the actual calibration factor to within two significant digits for this case.

To analyze confidence in the ability of the GA-optimized coupled model methodology to match a known solution, a Monte Carlo technique is used. The GA is run on the same problem 100 times with different initial random seeds. From the resulting sample of solutions we are able to estimate the mean, median, and error bars. Figure 14.7 depicts the mean calibration factor at each source as found by the GA along with the corresponding error bars. The inner error bars represent one standard deviation. The outer bars denote the 90% confidence interval; that is, 5% of the solutions are above the highest bar and 5% are below the lowest. We see that we are 90% confident that solutions range between 0.97 and 1.03 for each source, closely bracketing the true solution of 1.0. The mean of the 100 cases ranges between 0.9976 for source number 1 through 1.003 for source 11. Thus, the mean value computed from 100 runs is even more reliable than the already good solutions from a single GA coupled model run. Therefore, a single GA-optimized coupled model run is accurate to within 3% and the mean of 100 runs accurate to 0.3%.

Our prior work confirmed that these results are not unique to this prescribed configuration – either the specific calibration factors or the geometry (Haupt et al. 2006). When a spiral geometry, ranging in source-receptor distance of 250–1,750 m, was used instead, the results showed that the GA coupled model can correctly apportion the sources and that using the mean of the 100 Monte Carlo runs reduced the error.

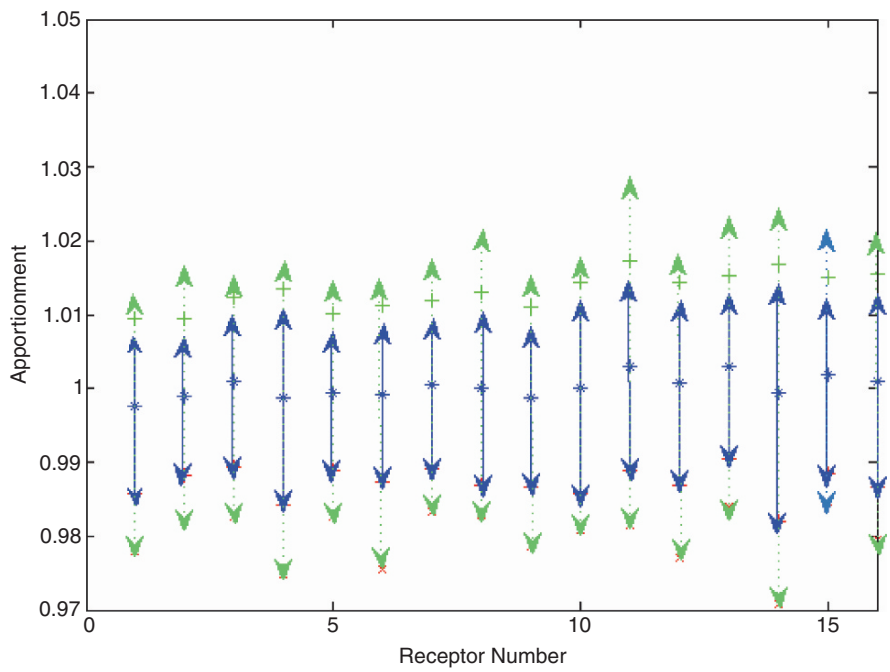


Fig. 14.7 Error bars from 100 Monte Carlo runs of the GA coupled model for the circular source configuration with correct apportionment factors all set to 1. The inner (solid) error bars are the one standard deviation level and the outer (dashed) error bars denote the 90% confidence level

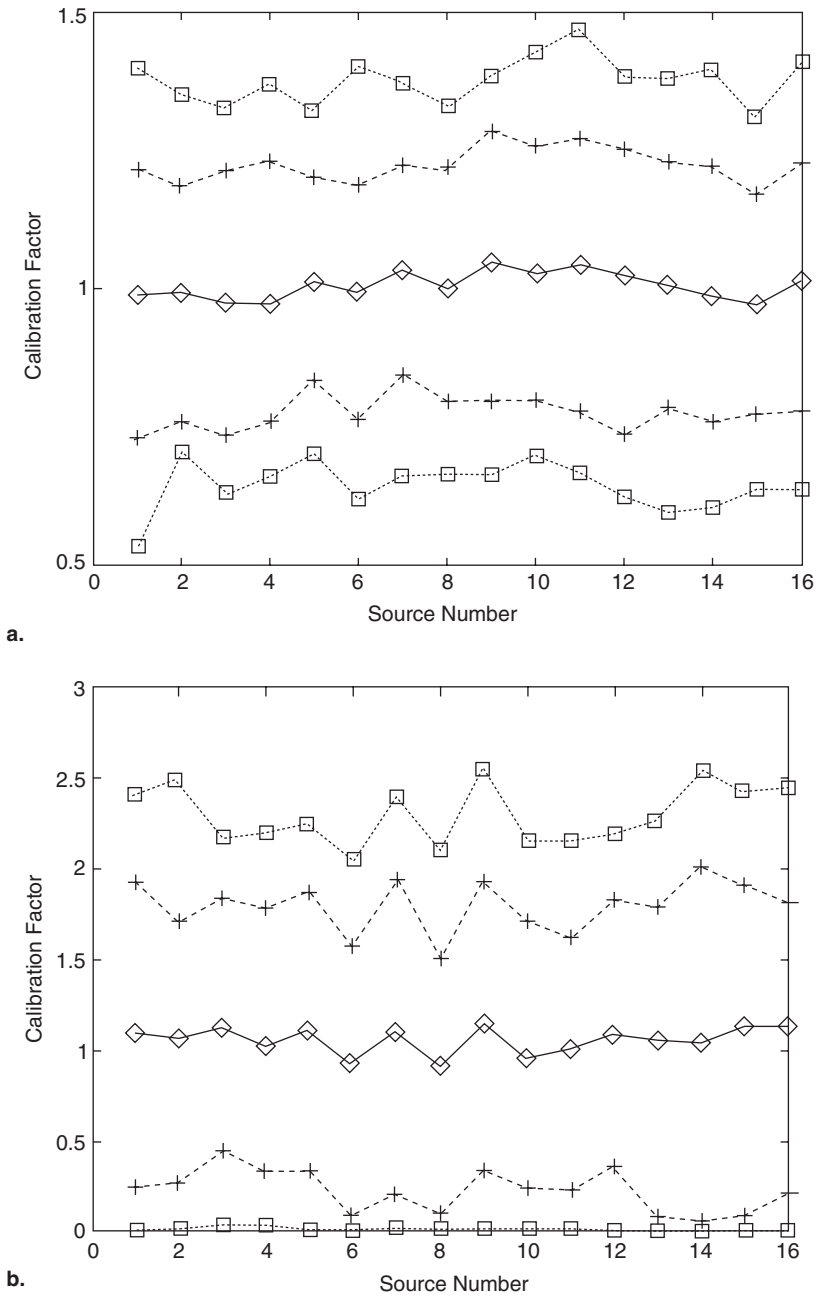


Fig. 14.8 Error bars from 100 Monte Carlo runs of the GA coupled model for the circular source configuration with correct apportionment factors all set to 1. White noise is added with amplitude equal to that of the signal: (a) Additive noise and (b) multiplicative noise. The inner solid black line marked by diamonds is the mean solution. Error lines denote one standard deviation (long dash marked by crosses) and 90% confidence level (short dashes marked by squares)

14.4.2 Analysis Including Noise

Real-world data do not have the pure signal available in our synthetically constructed data. Typical situations involve errors and uncertainties in both the emission and receptor data as well as the meteorological data. In addition, there is an inherent mismatch between the ensemble average nature of the model predictions and the single realizations yielded by the monitor measurements. In our analysis here, we simulate the aggregate uncertainty by incorporating white noise into the data, and then using the GA coupled model to optimize the calibration factor. No assumption is made of the source of this noise. It represents errors in both the monitored data and in the modeling process (ranging from uncertainty in source strength,

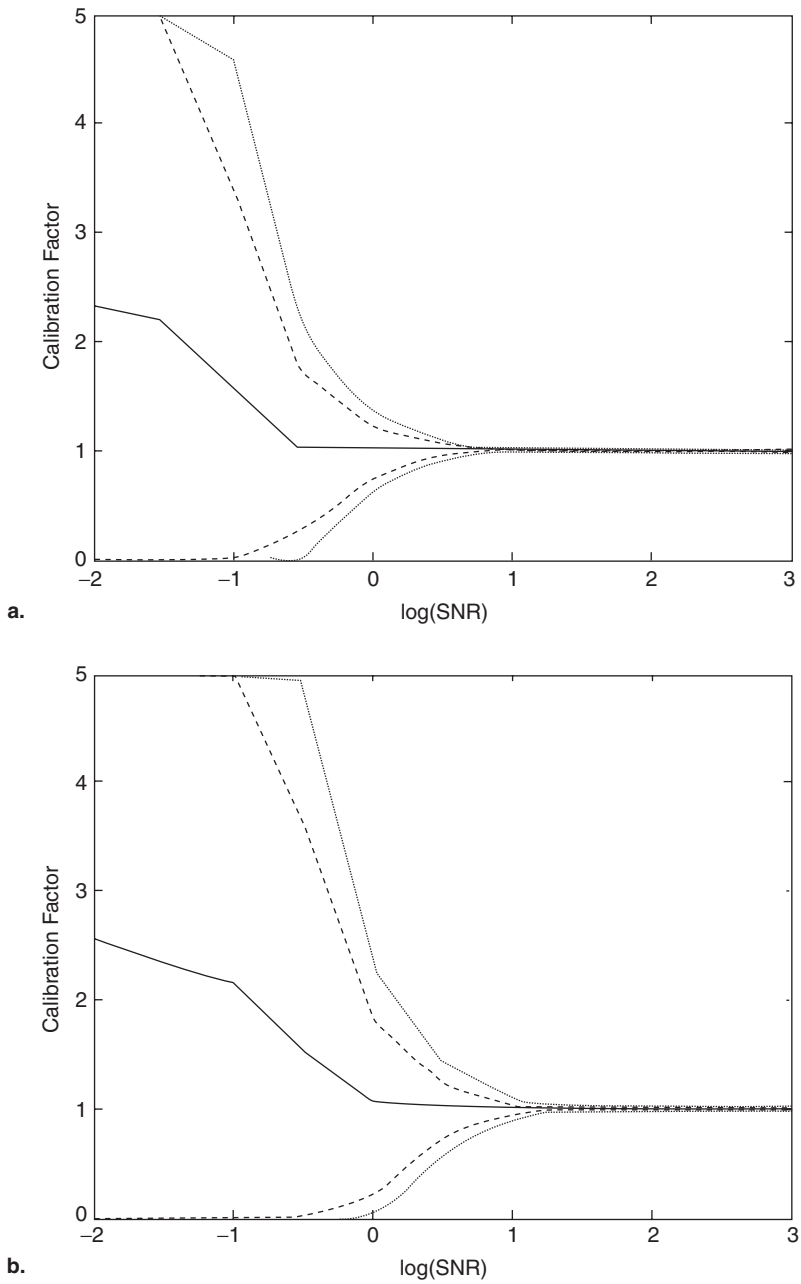


Fig. 14.9 Error bars from 100 Monte Carlo runs of the GA coupled model for the circular source configuration with correct apportionment factors all set to 1. White noise is added for various amplitudes as indicated on the abscissa: (a) Additive noise and (b) multiplicative noise. The inner solid black line is the mean solution. Error lines denote one standard deviation (long dash) and 90% confidence level (short dashes)

meteorological data, numerical error, and modeling simplifications, including the ensemble averaging assumption).

We again consider the circular geometry (see Section 14.3.1) with meteorological data representing 16 points of the wind rose. The Monte Carlo analysis uses assumed source calibration factors of all ones to create the receptor data. In this case, however, two separate methods of including white noise with an amplitude equal to that of the signal are used to simulate errors and uncertainties in the modeling process. First, white noise with mean amplitude of 1.0 is added to the dispersion model when creating the synthetic receptor data. The second analysis uses white noise to multiply the signal. Thus, the receptor data that goes into R_{mr} in equation (14.1) includes as much noise as signal. The GA coupled model is then used to compute the optimal calibration factors.

Figure 14.8 shows the mean, standard deviation, and 90% confidence interval for 100 runs of the GA coupled model. Figure 14.8a depicts additive noise while Fig. 14.8b depicts the multiplicative noise. In both cases, the curves depicting the error span a much wider range than for the case with no noise shown in Fig. 14.7. Aggregating the full 1,600 source cases (16 sources all with actual apportionment value of 1.0 over 100 runs) is equivalent to having 1,600 runs for a single source. Such an aggregation produces a mean value of 1.0066 for additive noise, quite close to the actual value of 1.0. The mean standard deviation of the aggregated 16 sources is 0.02595, an order of magnitude larger than for the case without noise (0.0015109) but still sufficiently small to provide confidence in model performance with imperfect information. Figure 14.8b indicates that the spread of the standard deviation and 90% confidence interval curves is greater for the multiplicative noise than for additive noise. In this case, the standard deviation for the multiplicative noise is 0.07449, which is greater because variability is proportional to the data itself.

Figure 14.9 depicts the performance of the coupled model over a range of signal to noise ratios (SNRs) for the additive and multiplicative noise cases. This plot aggregates the data over all 16 sources. We see that as long as $\log(\text{SNR}) > 1$, the solutions are quite close to the actual solution of 1.0 and the scatter is quite small. As noise becomes greater than the signal ($\log(\text{SNR}) < 0$), however, the computed solution diverges from the actual and the scatter becomes wider. Note that the mean of the solutions is still 1.0. At $\log(\text{SNR}) = 0$ the noise equals the signal and we have the case presented in Fig. 14.8 above. As expected, when the noise becomes much larger than the signal, as on the left side of the plot, the coupled model no longer reconstructs the solution reliably. In fact, the mean solution tends to 2.5, which is the center of the range allowed in the optimization routine. The standard deviation and 90% confidence lines approach the limits of the range. For multiplicative noise, the variability increases with decrease in SNR more rapidly than for the additive noise.

Haupt et al. (2006) report results for SNR analysis of other source configurations. The results described above generally hold and can be summarized as: (1) when multiple runs are averaged, confidence in the results is higher and (2) the GA coupled model run in Monte Carlo mode can apportion the sources correctly in the presence of noise of the same order of magnitude as the signal.

14.5 Tuning Meteorological Data

Accurate transport and dispersion modeling of pollutant releases requires accurate meteorological data – in particular, an accurate wind field. For most dispersion modeling applications, we don't have the meteorological fields at the preferred resolution, making precise computation of atmospheric dispersion quite difficult. Moreover, available wind data are not always accurate or representative. Thus, accurate source characterization can be difficult.

Here we present a new GA-based method that addresses the uncertainty associated with meteorological data by using a GA to tune the surface wind direction in addition to pollutant source characteristics. This method is an extension of the GA-coupled model described in Section 14.2. This extended method has an advantage over the original GA-coupled model in that it is far less sensitive to the uncertainty in meteorological data.

14.5.1 Architecture

Because this problem is different than the simpler source characterization problem presented earlier, changes must be made in the model architecture. As discussed in Section 14.2, the coupled model considers an array of candidate source locations, and the GA optimizes the strength of each candidate source by comparing dispersion model predictions with monitored receptor data. The source(s) with non-zero strengths are then assumed to be the actual emitters of the pollutant. In the method presented in this section, however, potential source locations are not known *a priori*. Neither is the wind direction. Instead, the source location comprises two of the GA-tunable parameters (x and y location) so that the model is free to choose any location within the domain. The wind direction can be any number between 0° and 360°. Thus, the performance of the method is not dependent on the appropriateness of a pre-defined candidate

source array or the presumed wind direction. Those are now free parameters, as is the source strength.

14.5.1.1 Forward Model

The disadvantage of this new architecture is that because the source locations and wind directions change as the GA evolves the population, the model must recalculate the pollutant dispersion from all candidate solutions at each iteration, thereby increasing the required CPU time. We use the Gaussian plume model (14.3) to test the method. This method uses concentration forecasts for each trial solution created using equation (14.3), receptor data for an arbitrary number of sites, and the GA to find the combination of source location, strength, and surface wind direction that provides the best match between the monitored receptor data and the expected concentrations.

14.5.1.2 Cost Function

The cost function used by the GA to evaluate each candidate solution is the root mean square difference between concentrations predicted by (14.2) and receptor data values, summed over all receptors. The cost function is similar to (14.2), except for changes in notation associated with the context of the current problem. Specifically, the cost function is defined as:

$$\text{Cost} = \frac{\sqrt{\sum_{r=1}^{TR} \left(\log_{10}(aC_r + 1) - \log_{10}(aR_r + 1)\right)^2}}{\sqrt{\sum_{r=1}^{TR} \left(\log_{10}(aR_r + 1)\right)^2}} \tag{14.10}$$

where C_r is downwind concentration at receptor r as calculated by (14.3), R_r is the receptor data value at receptor r, TR is the total number of receptors, and a is a constant. It is necessary to add 1.0 to the concentrations because the logarithm of zero is undefined. Doing this has the beneficial side effect of minimizing the contribution from the weakest concentrations values whose magnitudes are many orders of magnitude less than 1.0. The data must therefore be scaled since many of the C_r and R_r values are several orders of magnitude less than 1.0. The scaling factor, a, depends on the sum of all data values over all receptors:

$$a = \max\left(\frac{1}{\sum_{r=1}^{TR} R_r}, 1\right) \tag{14.11}$$

If the receptor data sums to a value greater than 1.0, then a is 1.0. Otherwise, a is greater than 1.0, so that at least some values are comparable in magnitude to 1.0. Scaling the concentration values allows the cost function to retain sensitivity to signal while still reducing sensitivity to noise via the logarithm.

14.5.1.3 Mating Scheme

In prior sections, the GA used a continuous version of single point crossover discussed in Chapter 5. For this reformulated problem, we obtain better GA performance by using a uniform crossover mating scheme that blends all parameters rather than just a single parameter. The uniform crossover method improves the average skill score (see Appendix for the definition of skill scores – lower skill scores are better) across six runs from 0.613 to 0.061, a remarkable improvement.

The superiority of this uniform crossover scheme to single-point crossover used in the models is most likely due to correlations between the effects of different parameters, specifically the dependence of plume structure on both wind direction and source location. Each wind direction has a unique optimal source location resulting in the best match to the receptor data. If the GA finds this location for a particular wind direction, and the wind direction is modified, the location is no longer optimal. Single-point crossover tends to converge to one of these "optimal" locations while failing to progressively improve the wind direction. Blending all parameters ensures that both the wind direction and source location are modified simultaneously, allowing both parameters to be progressively improved through the GA and decreasing the likelihood of premature convergence. Because of the correlations described above, the changes resulting from simultaneous modification of the wind direction and source location must complement each other. In a general sense, effects of parameters that are highly correlated in other applications are expected to exhibit similar behavior here.

14.5.1.4 GA Parameters

With the all-blending mating scheme, we have chosen to alter the GA parameters toward larger population sizes and smaller mutation rates. With a population size of 1,200, the GA can find the solution in a single run with 100 iterations or less. Larger population sizes than 1,200 and longer runs than 100 iterations result in slightly better performance, but the improvement is not significant when compared to the extra computing time. Smaller population sizes often converged too quickly to an incorrect solution, even when using a high mutation rate. Here, we use a mutation rate of 0.01 and a crossover rate of 0.5 for this problem of finding source location and wind direction in addition to source strength.

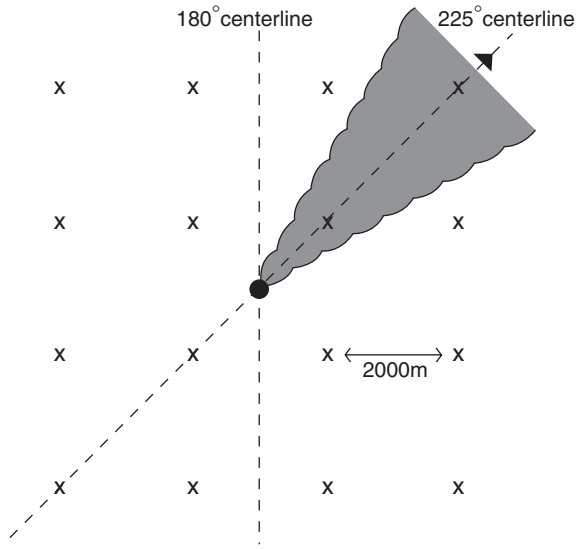


Fig. 14.10 The synthetic setup for a 4-by-4 grid of receptors. The black dot in the center represents the source and the X's are the receptors, each separated by 2,000 m. The dashed lines represent the plume centerline for the two wind directions considered in the synthetic data sets, and the shaded area represents a sample plume for the 225° wind direction

14.5.2 Demonstration

To demonstrate and validate the method of tuning meteorological data and source characteristics, we use synthetic data produced by (14.3) as receptor data. We place the receptors on a grid surrounding a single source with 2,000 m separating each receptor, and the source located in the center of the receptor domain at the point defined as the origin (0,0). To determine the dependence of model performance on the quantity of receptor data available, model runs are performed using 2-by-2, 4-by-4, 8-by-8, 16-by-16, and 32-by-32 grids of receptors. Synthetic data is produced for each receptor configuration for two different wind directions, 180° and 225°. These two wind directions represent opposite scenarios: a wind direction of 180° places the plume centerline directly between receptors, and a wind direction of 225° places the plume centerline directly over the receptors located along the $x = y$ diagonal in the northeast quadrant of the domain. Figure 14.10 shows the source and receptor setup for a 4-by-4 grid of receptors, where the black dot in the center is the source, the Xs are the receptors, and the dashed lines are the plume centerlines for the 180° and 225° wind directions. Model runs are performed for these five receptor configurations and two wind directions.

Table 14.4 shows the results for six different setups (receptor grids of 8-by-8, 16-by-16, and 32-by-32, for each of two wind directions) using a population size of 1,200 and 100 iterations. All GA runs produced a solution close to the actual, and some a tolerance of 0.01° in wind direction, 1% of source strength, and 1.0 m in source location. It may be puzzling at first that one of the 32-by-32 runs returned a worse result than

Table 14.4 GA-produced wind directions, source strengths, source locations, and skill scores for six synthetic configurations using a population size of 1,200, mutation rate of 0.01, after 100 iterations, for a single GA run. The correct solution is $\theta = (180°$ or $225°)$, strength = 1.00, and (x, y) = (0, 0). Appendix describe skill scores

Configuration	θ	Strength	(x, y) (in m)	Skill score
8-by-8, $\theta = 180°$	184.12°	2.96	−417, 1,346	1.4581
8-by-8, $\theta = 225°$	223.95°	1.06	−26, − 56	0.1952
16-by-16, $\theta = 180°$	180.01°	1.00	−1, 0	0.0029
16-by-16, $\theta = 225°$	225.01°	1.00	−1, 1	0.0019
32-by-32, $\theta = 180°$	180.00°	1.00	0, 0	0.0000
32-by-32, $\theta = 225°$	220.27°	1.12	−123, 519	0.6870

either of the 16-by-16 runs, but this occurred because each GA run begins with a random initialization, and the results in Table 14.4 reflect a single "test" run, not an average over many runs. The 32-by-32, 225° test run was just not as fortunate in its initialization. In general, runs with a 32-by-32 receptor grid perform at least as well as runs with fewer receptors.

Table 14.5 Number of runs (out of six) that produced a solution within tolerance for the given combination of population size and number of iterations. The rows are different population sizes, and the columns are different numbers of iterations

	Iter = 50	Iter = 100	Iter = 150	Iter = 200
Pop = 400	3	4	4	5
Pop = 800	4	4	4	5
Pop = 1,200	5	6	6	6
Pop = 1,600	5	6	6	6

14.5.2.1 Refinement

The solution after the 100th GA iteration is often close to, but not exactly at the global minimum of the cost function. Increasing the number of iterations above 100 does not greatly improve the solution for this reformulated problem. Therefore, we investigate whether a hybrid GA incorporating a traditional gradient descent method such as the Nelder-Mead Downhill Simplex NMDS method (Nelder and Mead 1965) could further improve the solution more efficiently than a GA does after the 100th iteration. The NMDS starts from a previously chosen point on a multidimensional surface (i.e. the cost function) and finds a local minimum in the vicinity of the starting point. For our application, we use the best GA-produced solution after 100 iterations as the starting point for the NMDS method. Gradient descent methods such as the NMDS are ineffective alone, however, as the it can only find the global minimum if the first guess is in the correct valley.

The NMDS method was run using each solution from Table 14.4. Each time, the NMDS returned a solution within our close tolerance limits, even for GA-generated starting points that were not "close enough". This improvement suggests that even though some of the specific values in the solutions from Table 14.4 are not within the tolerances, they are within the same cost function basin as the true solution. Under these circumstances, the NMDS can be used effectively to further improve the accuracy of the solution after the termination of the GA. The procedure as a whole is often called a hybrid GA, where a GA first is used to locate the basin of the global minimum of the cost function, and then the more traditional NMDS method is used to fine-tune the minimum. This hybrid GA produces a consistently good solution, better than either the GA or the NMDS method alone, in less computation time than the GA alone.

Running the GA beyond the 100th iteration does continue to improve the solution, but not as efficiently as the NMDS algorithm. Thus, we wish to run the GA just long enough to get to a solution that is an in-basin starting point for the simplex method. To determine where we should stop the GA, we ran the hybrid GA using 16 combinations of population size and number of iterations (each of which is proportional to computing time) to determine how much computing time is necessary to obtain an in-basin starting point.

Table 14.5 shows how many of six runs returned a solution within the tolerance after application of the NMDS method for each combination of population size and number of GA iterations. The combination of population size of 1,200 and 100 iterations was the most efficient to achieve this level of accuracy for all six runs made with the least computing time, the reason we use these values here.

Since the NMDS method is fairly efficient, could we just randomly generate initial guesses and still converge to the solution? Has the GA added any value? To answer these questions, we make multiple NMDS runs originating with random starting points within the solution domain. Table 14.6 indicates the number of function calls (a uniform unit of computing time) required by the GA and the random initialization NMDS method to find a solution within tolerance. The results were averaged over two runs for each receptor and wind direction configuration, for a total of 12 runs. The number of function calls required in any individual run using the NMDS varies greatly because the success of the NMDS method depends on the starting point, and the total number of function calls required is simply a function of how long it takes to produce an in-basin first guess. Because these first guesses are random in this experiment, it is not surprising that in some instances, the NMDS method found the solution faster than the GA. The performance of the GA, however, is far more consistent than NMDS over the

Table 14.6 Number of cost function evaluations required to find the solution for the GA and the Nelder-Mead downhill simplex, averaged over two runs for each configuration. Each configuration consists of an n-by-n receptor grid and a wind direction of either 180° or 225°

Configuration	GA function calls	Nelder-Mead function calls
8-by-8, $\theta = 180°$	19,200	17,180
8-by-8, $\theta = 225°$	1,200	123,225
16-by-16, $\theta = 180°$	13,800	60,874
16-by-16, $\theta = 225°$	3,600	121,035
32-by-32, $\theta = 180°$	10,200	16,996
32-by-32, $\theta = 225°$	23,400	133,034

12 runs performed, because it is able to overcome an unfortunate starting population and find the basin of the global minimum. Averaged across all six configurations tested, the GA took an average of 11,900 function calls to find a solution within tolerance, while the NMDS method took an average of 78,725 function calls. Thus, running the simplex from random starting points until the solution is found is inefficient compared to the GA, and particularly to the GA-NMDS hybrid. Moreover, if we did not know the correct solution *a priori*, the hybrid GA would assure us of convergence, particularly with multiple runs while the NMDS method alone would not.

14.5.2.2 Receptor Grid

How much receptor data is necessary to determine both wind direction and the source characteristics? The reason Tables 14.4 and 14.6 do not give results for the 2-by-2 and 4-by-4 receptor grids is that correct solutions could only be found consistently when using at least an 8-by-8 grid of receptors. For a 2-by-2 receptor grid, solutions were nearly random. For a 4-by-4 grid, solutions were somewhat better, but not nearly as good as the 8-by-8 grid solutions. This result suggests that a 4-by-4 grid of receptors does not provide enough receptor data to distinguish the effects of wind direction from those of source location and source strength. It does not imply that more than 16 total receptors are needed, as only two or three of the receptors in a 4-by-4 grid provide useful data (the others are outside the plume or nearly so). Because there are four parameters to be tuned (wind direction, source strength, and two for source location), having fewer than four data values does not provide enough information to resolve all the unknowns. In contrast, for an 8-by-8 grid, the number of receptors inside the plume exceeds the number of unknowns, so the hybrid GA is successful.

14.5.2.3 Noisy Observations

The success of the synthetic data runs is partly due to the exact match between the synthetic receptor data and the expected concentrations calculated by (14.3). This was also the case with synthetic data in Section 14.3 with the GA-coupled model. Therefore, we again contaminate our synthetic data with white noise to simulate the variability and errors present in monitored receptor data in order to gauge our new hybrid GA's performance when faced with inexact receptor data.

Twelve model runs are performed for each combination of receptor grid size and signal-to-noise ratio (SNR). Six SNRs are tested: infinity (no noise), 100, 10, 1, 0.1, and 0.01. Analyses are made for wind directions of both 180° and 225°. It is expected that runs with more receptors are less sensitive to noise than runs with fewer receptors. While an 8-by-8 receptor grid provides sufficient information to produce the solution with no noise, it may not provide enough information when degraded by noise.

Figure 14.11 shows median skill scores across twelve runs for each combination of SNR and n-by-n receptor grid. Recall that lower skill scores denote better solutions as described in Appendix. The median is used instead of the mean, because the median is less sensitive to outliers and is more indicative of what to expect in a single run. Figure 14.11 shows the results for multiplicative noise; results for additive noise are similar. The figure shows that the ability of the model to compute the correct solution is not significantly affected as long as the magnitude of the signal is greater than the magnitude of the noise (i.e. SNR > 1). For SNR $= 1$ where the signal and noise are of equal magnitude, the model performs slightly better with more data beyond an 8-by-8 grid. Performance at this point has deteriorated, however, as indicated by the sharp skill score gradient between SNR $= 10$ and SNR $= 1$. For runs with more noise than signal (SNR < 1), the GA is unable to compute the solution to any acceptable degree of accuracy.

Recall that in the synthetic data runs with no noise, the NMDS algorithm can further improve the

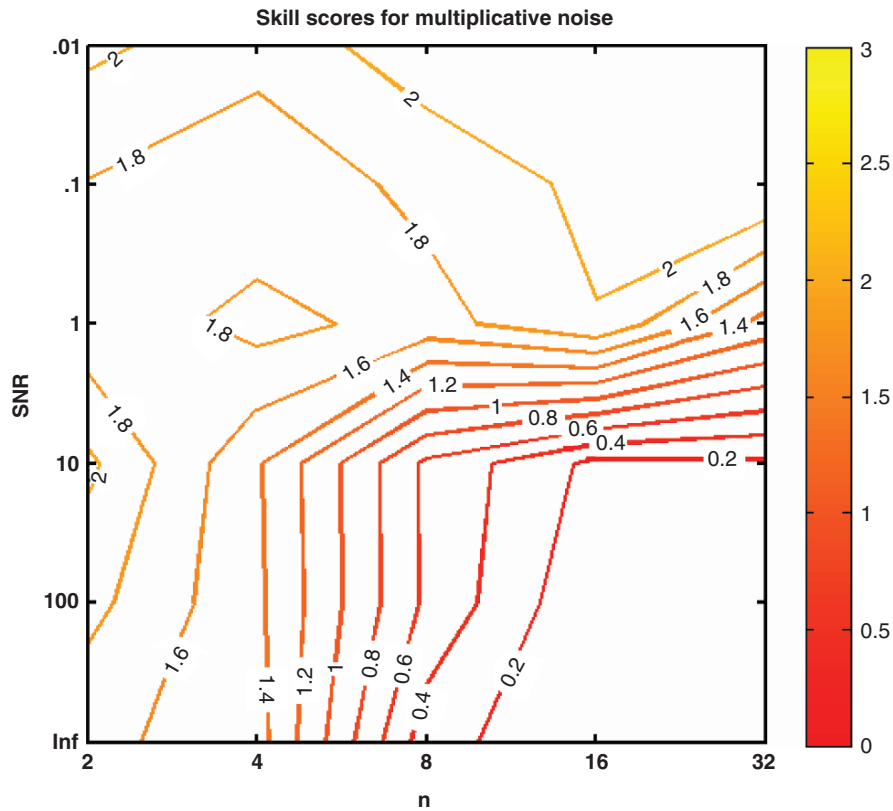


Fig. 14.11 Contour plot of median skill score for various n-by-n receptor grids and signal-to-noise ratios (SNRs) for multiplicative noise. The median skill scores are taken over 12 runs. Lower skill scores denote better solutions

solution found after the 100th GA iteration. In the runs with noise, however, application of the NMDS algorithm after the 100th GA iteration did not appreciably improve the solution. The average skill score of the GA-produced solutions across all SNRs and receptor grids was 1.578, while the average skill score after the application of the Nelder-Mead downhill simplex was 1.582, which is slightly worse. This result is not surprising, because after the receptor data is contaminated with noise, the solution corresponding to the lowest cost function value is usually not the correct solution. While the NMDS method may find a lower cost function value than the GA, the objective skill score does not consider the cost function value, only the specific values of each parameter.

14.5.3 A Parting Look

To help cope with the uncertainty in meteorological data, we have described a method that tunes wind direction and contaminant source characterization simultaneously by using a GA. The model works extremely well for synthetic data given a grid of at least 8-by-8 receptors. A smaller set of receptors, such as a 4-by-4 grid, does not provide enough data to distinguish wind direction from source characteristics. Using synthetic data contaminated with white noise, as long as the magnitude of the noise does not exceed the magnitude of the signal, the GA can still find the wind direction, source location, and source strength fairly well. Increasing the quantity of available data increases the amount of noise the GA can cope with in determining the approximate solution.

For demonstration purposes, the meteorological tuning experiments presented here use a Gaussian plume equation to calculate expected downwind concentrations in order to reduce the computational complexity. For a real data application, a more sophisticated dispersion model can provide a closer match to the receptor data than the Gaussian plume equation. The current model configuration, however, requires a new set of dispersion calculations in each GA iteration, so the direct use of a more complex model would impose substantial computational cost.

14.6 Incorporating Realism: SCIPUFF and Field Test Data

We have shown that the GA-coupled model can determine the source characteristics for pollutant emissions using synthetic data produced by the Gaussian plume equation. We now wish to use the coupled model as a source characterization tool in the context of an operational dispersion model and real data. Thus, we replace the Gaussian plume equation (14.3) with a more sophisticated dispersion model, SCIPUFF. This new coupled model can then be tested with real contaminant data. If successful, such a coupled model could be useful in determining the source characteristics for those hazardous release events where monitored contaminant concentrations are available. For this section, we also assume that the meteorological data are known. The general architecture of the GA coupled model of this section returns to that described in Section 14.2.

14.6.1 Adding SCIPUFF as the Dispersion Model

The primary upgrade to the GA coupled model of Section 14.2 is the replacement of the Gaussian plume equation (14.3) with the much more sophisticated SCIPUFF dispersion model (Sykes et al. 1998). As the forward component of the coupled model, SCIPUFF calculates the contributions from each potential source. These contributions are represented by matrix **C** in (14.1).

SCIPUFF, the *S*econd-order *C*losure *I*ntegrated *PUFF* model, is an ensemble average transport and dispersion model that computes the field of expected concentrations resulting from one or more sources at multiple times. The model solves the transport equations using a second-order closure scheme, and treats releases as a collection of Gaussian puffs (Sykes et al. 1986; Sykes and Gabruk 1997). SCIPUFF can be used for dispersion applications requiring expected concentrations of source material. SCIPUFF is a suitable choice for insertion in our GA coupled model because of its ability to compute expected concentrations over predefined time periods for any number of sources, and the ease in integrating its output into matrix **C** of (14.1).

SCIPUFF is run once for each potential source considered by the coupled model. The output from each SCIPUFF run corresponds to a particular column in the **C** matrix. This use of SCIPUFF does not impose substantial computational cost, because the SCIPUFF runs only need to be executed once prior to the GA initialization and not in every GA iteration.

The parameters of the GA return to those of Section 14.2, with a population size of 8, mutation rate of 0.2, crossover rate of 0.5, and the same cost function (14.2).

14.6.1.1 Validation with SCIPUFF

To gauge the impact of upgrading the GA coupled model's forward component, the validation technique performed in Section 14.4 using the Gaussian plume equation is repeated for the coupled model incorporating SCIPUFF. The validation consists of model runs using synthetic data produced by SCIPUFF. Subsequent tests, including validation with real data, can then be performed with confidence that any issues encountered are not related to incorporating SCIPUFF into the coupled model.

For the validation phase, SCIPUFF was used to create synthetic receptor data, representing matrix **R** in (14.1). The synthetic receptor data are instantaneous contaminant concentrations at a previously defined receptor location 5 m above the surface, with each time-dependent observation corresponding to one value of **R**. Sets of synthetic data corresponding to particular source configurations were created using a synthetic two-dimensional wind field using the same circular geometry used in Section 14.3.1, with 32 independent meteorological periods (here, hours) and 16 potential sources. The validation runs also use the same synthetic meteorological data as in Section 14.3.1. The validation uses a 100-run Monte Carlo simulation for each set of synthetic data as done in Section 14.4. Three source configurations were analyzed with similar results; this section focuses on a spiral configuration with varying source strength ($\mathbf{S} = [0, 1, 2, 3, \ldots, 0, 1, 2, 3]^T$). Allen et al. (2006) provides results for the other source configurations along with more detailed analysis.

Table 14.7 shows the means and standard deviations for 4 of the 16 sources, each corresponding to a different **S** value (0, 1, 2, or 3). All of the means are very

Table 14.7 Means and standard deviations for four sources in the spiral configuration setup across 100 Monte Carlo runs

	Source 1	Source 6	Source 11	Source 16
Calibration factor	0	1	2	3
Mean	0.0142	0.9996	2.0004	2.9998
Std Dev	0.0145	0.0142	0.0193	0.0167

close to the known solutions (with the slight exception of source 1), and all of the standard deviations are less than 0.02. The mean for source 1 is further from the solution than for the other sources because the GA imposes a lower bound of 0 on the solutions. Overall, the GA does an exceptional job of approaching the solution, not just in terms of the mean across all 100 Monte Carlo runs, but also for single runs, as shown by the small values of the standard deviations.

As in Section 14.4, additional Monte Carlo simulations were run using synthetic data contaminated with noise. The noise simulates the impact of imprecise monitoring data, errors in the meteorological data, and the disparity between the ensemble average nature of the model as compared to data from a specific realization. Figure 14.12 summarizes the results for the spiral configuration using multiplicative noise. The figure shows the GA-computed **S** for 4 of the 16 sources as a function of the logarithm of the signal-to-noise ratio. These four sources are representative of each of the four different solutions. Dashed error bars signify plus and minus one standard deviation from the mean, and the dotted error bars represent the 90% confidence interval. A detailed discussion of the results can be found in Allen et al. (2007).

The graphs and results from other source configurations (not shown) are quite similar. This suggests that the choice of dispersion model used within the coupled model does not affect the performance of the GA in obtaining the optimal solution, allowing models of increasing complexity to be used in the coupled model with no performance-related side effects. Computing time depends more on the GA than the dispersion model, because the dispersion model is only run once for each source, further supporting the use of a dispersion model of any level of complexity within the coupled model. Of course, this does not mean

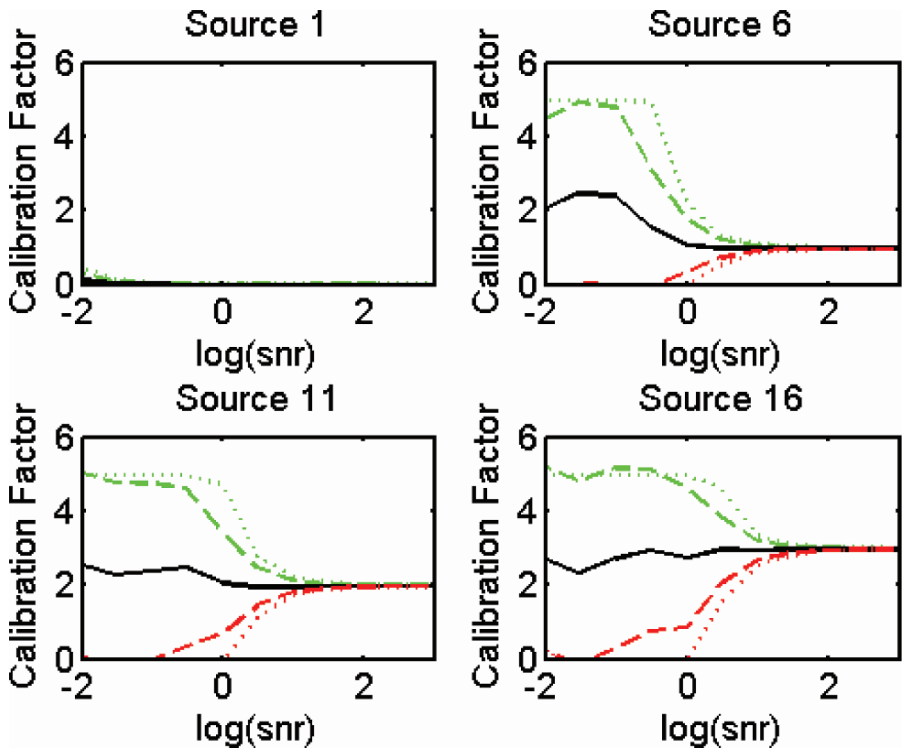


Fig. 14.12 Calibration factor as a function of the signal-to-noise ratio (SNR) for the spiral source configuration using multiplicative noise. Four sources with different S values are shown: Source 1 (S = 0), Source 6 (S = 1), Source 11 (S = 2), and Source 16 (S = 3). Mean (solid), standard deviation (dashed), and 90% confidence interval (dotted) are shown

incorporating SCIPUFF into the coupled model does not upgrade the performance in a general sense, but only that it does not increase the programming complexity or hinder the performance of the GA.

14.6.2 Verification with Monitored Data

Now that we have validated the GA coupled model incorporating SCIPUFF, we can conduct coupled model runs using real data – specifically, neutrally buoyant tracer concentration data from the Dipole Pride 26 (DP26) field tests. These runs are used to demonstrate the model's ability to characterize pollutant sources correctly despite the stochastic scatter of realizations around the forecast ensemble mean.

The DP26 field experiments took place in November 1996 at the Nevada Test Site (Biltoft 1998). The tests released sulfur hexafluoride (SF6), a passive tracer, at locations nearby a domain of 90 receptors. Seventeen different field tests were carried out during the DP26 experiments. Our study only used data from 14 of these tests due to missing data in the other three tests. Figure 14.13 shows the test domain and the orientation of sources and receptors. N2, N3, S2, and S3 are the source locations, and the thick black lines show the approximate receptor locations. Further details on these field experiments can be found in Biltoft (1998) and Watson et al. (1998).

Chang et al. (2003) used the DP26 data to validate various dispersion models, including SCIPUFF. While SCIPUFF performed as well as the other dispersion models they tested, about 50–60% of SCIPUFF-predicted concentrations came within a factor of two of the observations. Most large errors occurred when the modeled puff missed the receptors altogether due to errors in the wind field. To alleviate the effects of these large errors and other issues associated with real data, several changes need to be made to the coupled model architecture.

In order to use data from all 90 receptors, the **C** and **R** matrices in (14.1) are expanded so that r > 1. Because the purpose of the GA-coupled model is to find a single source apportionment vector, **S** providing the best fit across all receptors, the calibration vector **S** remains a one dimensional vector. If the model matches the data perfectly, a single **S** vector would be all 1.0s.

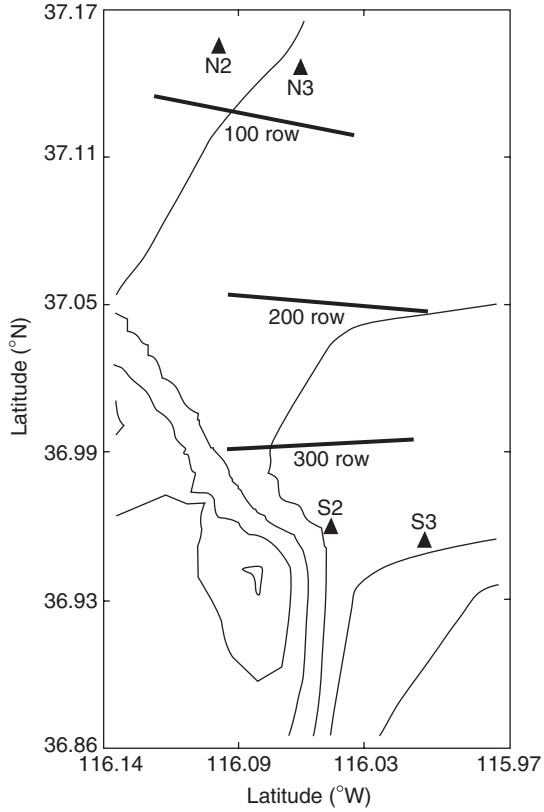


Fig. 14.13 Dipole Pride 26 test domain as represented in coupled model. N2, N3, S2, and S3 are the emission source locations. The thick black lines represent the approximate locations of the receptors (30 along each line). The thin black lines represent terrain contours, corresponding to heights of 1,000–1,519 m. Source: Modeled after similar figures in Biltoft (1998) and Chang et al. (2003).

Two modifications also must be made to the cost function (14.2) – due to the large errors in contaminant magnitude often found in real data applications. First, the cost function incorporates the natural logarithm of the squares of differences as in Section 14.5.

$$\mathrm{RMS} = \frac{\sum_{r=1}^{90}\sqrt{\sum_{m=1}^{M}\left(\log_{10}\left(C_{mnr}\cdot S_n+1\right)-\log_{10}\left(R_{mr}+1\right)\right)^2}}{\sum_{r=1}^{90}\sqrt{\sum_{m=1}^{M}\left(\log_{10}\left(R_{mr}+1\right)\right)^2}} \tag{14.12}$$

Second, it is necessary to add 1.0 to the concentration values before taking the logarithm because

the logarithm of zero is undefined. For this application, the concentration values are typically several orders of magnitude greater than 1.0, so the values are not dwarfed. Section 14.5 discusses a scheme for applications where most concentration values are less than 1.0.

Allen et al. (2007) shows that the logarithmic cost function (14.12) is more effective in determining the source characteristics than the linear cost function (14.2), particularly for finding the source location and emission time of an instantaneous release. This logarithmic variable transformation acts to ameliorate the order of magnitude differences that often arise in concentration data. This behavior is desired because the primary issue in source identification is not strictly the magnitude, but rather the non-zero nature of each source's contribution (i.e. whether or not a source's puff passes over a particular receptor at all). For example, if the receptor data value is 200 parts per trillion (ppt), but the model's predicted concentration is 2,000 ppt, a logarithmic cost function rates the value of 2,000 ppt more highly than a value near 0.0 ppt.

Haupt et al. (2006) and Section 14.3.3 above show that the RMS cost function produced the most efficient convergence. Therefore, all cost functions considered here involve some form of a squared difference. The two normalization schemes discussed here affect model performance, but the specific normalization values used are arbitrary, because the GA mating mechanism is based on ranking rather than absolute difference.

One more issue with real data applications such as DP2 is that it is difficult to characterize non-emitting sources whose potential plumes disperse completely outside the receptor domain. For instance, some potential source may be downwind of the receptors. We deal with this issue by introducing a scale factor that adjusts each source's maximum allowed magnitude. This scale factor sums each column in the **C** matrix representing the pollutant contribution of each source n, and normalizes that sum by the maximum contribution from any source to produce a number ranging from 0.0 to 1.0.

$$\text{scale}(n) = \frac{\sum_{r=1}^{90}\sum_{m=1}^{M} C_{mnr}}{\max\left(\sum_{r=1}^{90}\sum_{m=1}^{M} C_{mnr}\right)} \tag{14.13}$$

The scale factor (14.13) is then multiplied by a predefined upper limit to give the maximum source strength allowed by the GA for each source. Sources that cannot emit into the domain have scale factors of 0.0, forcing the GA to limit these sources' $\mathbf{S}_n$ values to 0.0. This method does not assume any prior knowledge regarding which sources are potential emitters, but does provide objective estimates of each source's potential contribution to the domain. This process eliminates the 50% of the candidate sources that are downwind of the receptor for the Dipole Pride data set.

A possible side effect of using the scale factor is limiting the maximum allowable strength for the correct sources below their actual strengths. To account for this side effect, the range of strengths allowed by the GA should be set beyond the expected range of possible strengths. The range should not be made too large, however, since the run-to-run variability in solutions is proportional to this range. Thus, **S** values are set to range from 0 to 10, increased beyond the original range of 0 to 5.

Several initial runs were made with the GA coupled model using the DP26 data and these coupled model modifications. The goal of these runs was to characterize the emission locations and times (strength characterization is the focus of subsequent sections). These runs used the four emission locations (N2, N3, S2, S3) at two times each, for a total of eight sources. S_n should be equal to 1.0 at the emitting sources (one or two sources in each field test), and 0.0 for all non-emitters, if all else is perfect. In other words, it should detect which source was the actual emitter for each field. In the initial runs, the correct source and time of emission were identified 64% of the time.

14.6.3 Performance Optimization

Now we seek to optimize the performance of the coupled model with the DP26 data set by performing various tests, each designed to determine the impact of different parameters. While the optimization is specific to DP26, many of the results can be applied to the coupled model in general for other data sets.

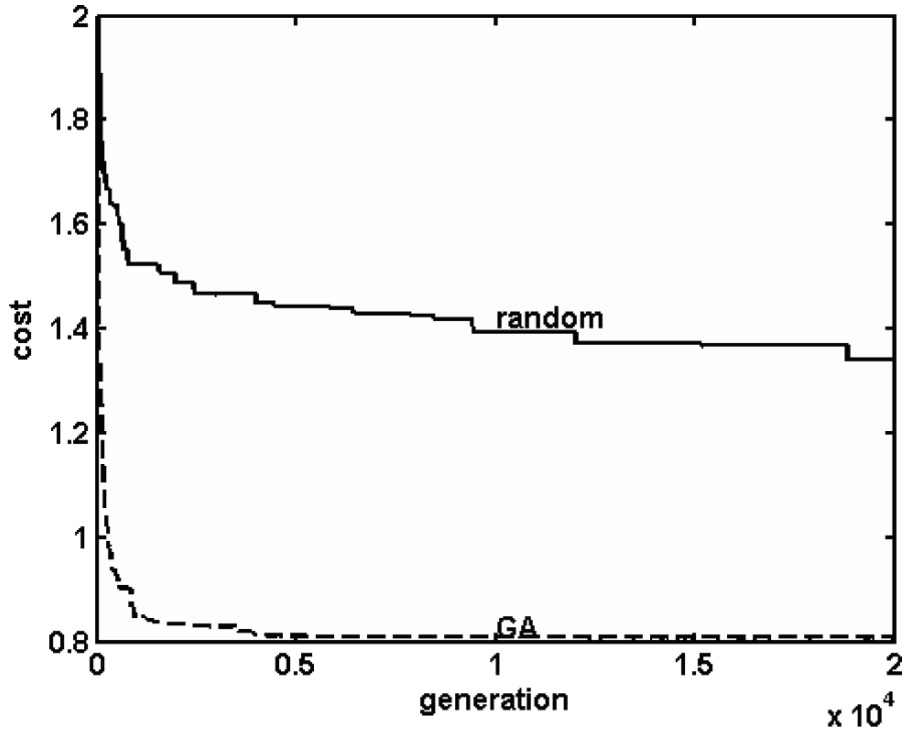


Fig. 14.14 Minimum cost function value as a function of iteration number for the GA (dashed) versus a random search method (solid), carried out to 20,000 iterations

14.6.3.1 GA vs. Random Search

The first test determines if solving the matrix problem requires the GA at all. The GA's performance is compared to the performance of the random search method in Fig. 14.14, which shows the minimum cost for one of the DP26 tests, as found by the GA (dashed) and the random search (solid), averaged over five runs, each with 20,000 iterations. While the "number of iterations" is specific to the GA, the corresponding computing time for the random search method is normalized to be equivalent to the number of GA iterations, so that the graph provides a fair comparison. The random search clearly took much longer to find a solution with a sufficiently low cost function value. In fact, out to 20,000 iterations, the random search never caught up to the GA while the GA converged to the optimal solution in about 7,000 iterations. This result shows that a random search is inefficient, and that a more sophisticated optimization method such as a GA is required.

14.6.3.2 Population Sizes and Mutation Rates

Section 14.3.4 presented a sensitivity study on GA population sizes and mutation rates using synthetic data and found that two combinations of sizes and rates were most efficient in converging to the solution: high population sizes coupled with relatively low mutation rates, and low population sizes coupled with high mutation rates. To determine if the same conclusion applies to a real-data application, a similar sensitivity study is made using the DP26 data set using 5 of the 14 field tests. The goal is to determine which combination of population size and mutation rate minimizes the number of cost function evaluations required for convergence to a correct solution.

Figure 14.15 shows the number of cost function calls required for 80 combinations of population sizes and mutation rates, averaged across five runs for each field test. The optimal mutation rate was 0.15, and the optimal population size was in the range of 4 to 12, similar to the results in Section 14.3.4. Unlike the

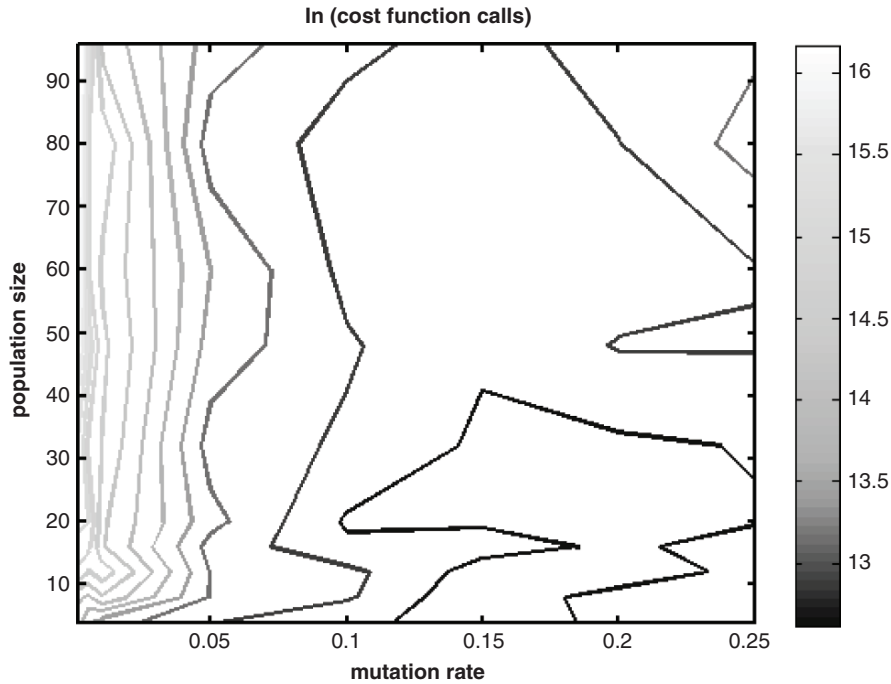


Fig. 14.15 Contour plot of average number of cost function calls versus mutation rate and population size for the coupled model. Darker contours correspond to fewer cost function calls. The number of cost function calls has been normalized by the natural log for viewing purposes

case in Section 14.3.4, however, a high population size coupled with a low mutation rate was not efficient in finding the solution. A possible reason for this difference is that these runs were conducted with a candidate source array of size 8 (N2, N3, S2, and S3 at two times each). The scale factor eliminates four of the sources, leaving four. With only four significant GA parameters in the chromosome, mating is less effective than mutation for finding progressively better solutions. Therefore, a relatively high mutation rate coupled with a rather small population size works well for this specific application.

14.6.3.3 Other Studies and Multi-stage Process

Other sensitivity studies were performed, resulting in the following conclusions, which are elaborated on in Allen et al. (2007) and Allen (2006):

- The DP26 data set provides receptor data every 15 min. SCIPUFF can also output values every 15 min; however, the DP26 receptor data are not instantaneous concentrations, but rather time-integrated averages. Shortening the output interval in SCIPUFF to 5 min and then averaging back up to 15 min improves the GA's performance; shortening the output interval beyond 5 min did not further improve solution accuracy.
- It is necessary to include all 90 receptors in the analysis to have the best solution accuracy. Using fewer than 90 receptors improves computing time, but at the expense of less accurate solutions.
- DP26 includes upper-air meteorological data, but the upper-air data was found to have little effect on the source characterizations that are based on surface data only.
- The run-to-run variability in solutions is proportional to the range of values allowed by the GA. As discussed earlier, this range should be larger than the range of all possible strengths because of the scale factor. Because correct sources had scale factors as low as 0.1 in some instances, the range of values allowed by the GA should be an order of magnitude larger than the range of presupposed possible strengths.
- In a typical model run, the source strength is underestimated because the GA attributes small amounts of pollutant to non-emitting sources, thus decreasing the calculated strength at the correct source.

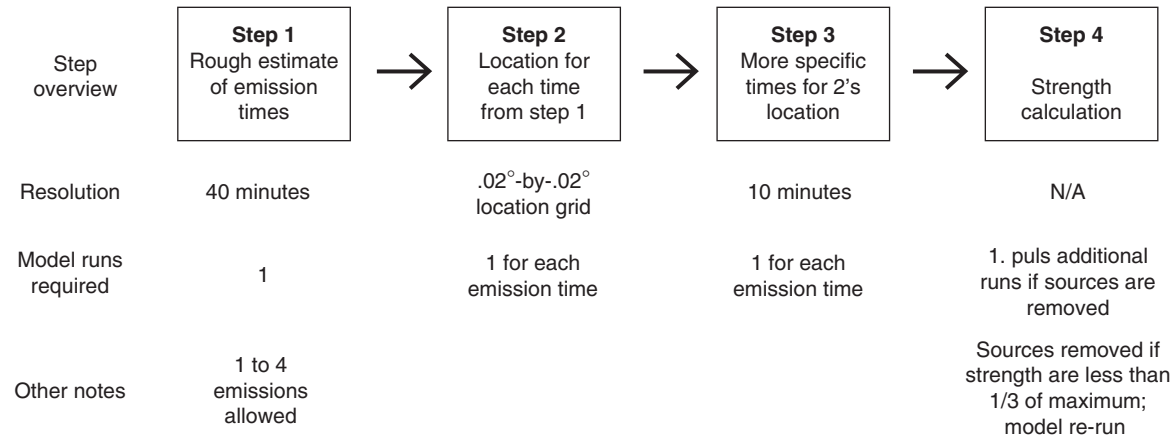


Fig. 14.16 Illustration of the steps in the multi-stage process for source characterization

A more accurate source strength is obtained in a model run by including only the correct source(s) in the candidate source array, thus forcing the GA to attribute all pollutant to only the correct source(s).

These observations helped us to produce an automated multi-stage process that best characterizes the location, time, and strength of the pollutant source(s) assuming as little *a priori* information about the sources as possible. Figure 14.16 provides an outline of the multi-stage process. The first stage uses a coarse grid designed to estimate the number of emission sources and time of each emission. The second stage performs a separate coupled model run for each emission time found in the first iteration. The third stage pinpoints the most probable emission time for each release by running the coupled model once for each emission with the locations found in the second stage in the source array. The final stage then calculates the strength of each emission. The final result is a list of emission locations, times, and strengths.

14.7 Summary and Prospects

This chapter has followed the development of an AI technique to solve a real world problem. We are attempting to identify and characterize a source of contaminant in spite of imprecise knowledge of source location, emission rate, and time of release; uncertain and changing meteorological conditions; monitoring errors; and the inherent uncertainty of turbulent transport and dispersion. In spite of these formidable problems, our GA coupled model is shown to work rather well. It was developed and tested using synthetic identical twin experiments and contrived geometries to test the limits and tune the method (Section 14.3.1). A realistic geometry (Section 14.3.2) revealed some additional limitations along with the successes. We saw that certain geometries could produce ill-posed problems for that model formulation. We also studied GA performance by varying the GA parameters of population size and mutation rate. For this complicated cost surface, the mutation operator is critical for GA convergence.

Next, a statistical analysis of model performance using Monte Carlo runs revealed that the average of multiple runs produces an even better solution. In spite of applying either additive or multiplication white noise to simulate the highly uncertain model environment, the GA coupled model is successful in the Monte Carlo framework, even when the noise level is of the same magnitude as the signal. As noise overshadows the signal, however, confidence in the solution degrades.

What about situations where either no meteorological data are available or the wind data are not representative of local conditions? Section 14.5 demonstrates an extended GA model to simultaneously search for source location, emission rate, and wind direction. A different mating scheme is required for this application, which leads to quite different choices for GA parameters of population size and mutation rate. This GA application combined with a more traditional

NMDS method speeds convergence once the GA has found the correct solution basin for the simplex starting point. Note that this traditional gradient-based method did not work well without the GA to provide that first guess.

These initial demonstrations were done in the context of a very basic Gaussian plume T&D model. To incorporate more realism into the dispersion process, Section 14.6.1 replaced the Gaussian plume with the refined SCIPUFF model. The GA coupled model can be validated much the same as the original version. The most realistic test was accomplished on data from the Dipole Pride 26 field test with all its inherent errors and uncertainties. Note that prior modeling studies had difficulty matching the T&D for these data. The GA model still showed success on some of the trials.

Subsequent work has used similar techniques to back-calculate up to seven modeling parameters: two-dimension location, emission height, source strength, time of release, wind direction, and wind speed (Long et al. 2008; Long 2007). A mixed integer genetic algorithm is able to characterize atmospheric stability (Haupt et al. 2008).

The GA coupled model is not perfect. Neither is any other model attempting to solve this difficult source characterization problem. The exercise does demonstrate, however, that an AI-based technique is competitive for solving a real world environmental problem.

Acknowledgements We thank Joseph Chang for providing the Dipole Pride 26 data, Ian Sykes for helpful information on SCIPUFF, and Kerrie Long for many helpful suggestions.

Appendix: Skill Scores

For the evaluation of results in the simultaneous tuning of surface wind direction and source characterization, an objective skill score is required to evaluate the proximity of solutions to the actual solution. The skill score used here is designed to weight the error in wind direction, source strength, and source location equally. The errors in each parameter are normalized to a [0,1] scale, with a score of 0 given to exact solution, and a score of 1 when inaccuracy exceeds a predefined upper bound. These scores are then added up to give a final score from 0 to 3, with a score of 0 for an exact solution.

The formulas for the three skill score components are:

$$S_{wind} = \ln\left(|\theta_{GA} - \theta_{act}| + 1\right)/5.199 \tag{14.14}$$

$$S_{str} = \max\left(\left(\frac{S_{GA}}{4 * S_{act}} - \frac{1}{4}\right), \left(\frac{S_{act}}{4 * S_{GA}} - \frac{1}{4}\right)\right) \tag{14.15}$$

$$S_{loc} = 1.0746 * \left(-\exp\left(-dist/1500\right) + 1\right) \tag{14.16}$$

where θ_{GA} is the wind direction found by the GA, θ_{act} is the actual wind direction, S_{GA} is the source strength found by the GA, S_{act} is the actual source strength, and *dist* is the distance from the GA-computed source location to the actual source location in meters.

The constants in these equations were computed to give the desired scores for various solutions – specifically, to give a score of 0 for an exact solution, and a score of 1 for a solution at or above a predefined threshold (180° for wind direction, a factor of 5 for the source strength, and 4,000 m for the source location). For example, in the wind direction equation (14.14), *ln*(181) is approximately equal to 5.199, so the constant 5.199 results in a score of 1 for the highest possible error of 180°. For each equation, if the computed value exceeds 1, the value is truncated to 1. The final skill score is $S_{wind} + S_{str} + S_{loc}$, where 0 is a perfect score, and 3 is the worst possible score.

References

Allen, C. T. (2006). *Source characterization and meteorological data optimization with a genetic algorithm-coupled dispersion/backward model*. M.S. thesis, Department of Meteorology, The Pennsylvania State University, p. 69.

Allen, C. T., Haupt, S. E., & Young, G. S. (2007). Source characterization with a receptor/dispersion model coupled with a genetic algorithm. *Journal of Applied Meteorology and Climatology*, *46*, 273–287.

Allen, C. T., Young, G. S., & Haupt, S. E. (2006). Improving pollutant source characterization by optimizing meteorological data with a genetic algorithm. *Atmospheric Environment*, *41*, 2283–2289.

Beychok, M. R. (1994). *Fundamentals of stack gas dispersion* (3rd ed., p. 193). Irvine, CA: Milton Beychok.

Biltoft, C. A. (1998). Dipole Pride 26: Phase II of Defense Special Weapons Agency transport and dispersion model validation. DPG Doc. DPG-FR-98-001, prepared for Defense Threat Reduction Agency by Meteorology and Obscurants

Divisions, West Desert Test Center, U.S. Army Dugway Proving Ground, Dugway, UT, 76 pp.

Camelli, F., & Lohner, R. (2004). Assessing maximum possible damage for contaminant release event. *Engineering Computations*, *21*, 748–760.

Cartwright, H. M., & Harris, S. P. (1993). Analysis of the distribution of airborne pollution using GAs. *Atmospheric Environment*, *27A*, 1783–1791.

Chang, J. C., Franzese, P., Chayantrakom, K., & Hanna, S. R. (2003). Evaluations of CALPUFF, HPAC, and VLSTRACK with two mesoscale field datasets. *Journal of Applied Meteorology*, *42*, 453–466.

Daley, R. (1991). *Atmosphericdata analysis*. Cambridge, UK: Cambridge University Press.

Haupt, R. L., & Haupt, S. E. (2004). *Practical genetic algorithms* (2nd ed., p. 255). New York: Wiley, with CD.

Haupt, S. E. (2005). A demonstration of coupled receptor/dispersion modeling with a genetic algorithm. *Atmospheric Environment*, *39*, 7181–7189.

Haupt, S. E., Young, G. S., & Allen, C. T. (2006). Validation of a receptor/dispersion model coupled with a genetic algorithm using synthetic data. *Journal of Applied Meteorology and Climatology*, *45*, 476–490.

Haupt, S. E., Haupt, R. L., & Young, G. S. (2008). A mixed integer genetic algorithm used in Chem-Bio defense applications, submitted to *Journal of Soft Computing*.

Kumar, A. V., Patil, R. S., & Nambi, K. S. V. (2004). A composite receptor and dispersion model approach for estimation of effective emission factors for vehicles. *Atmospheric Environment*, *38*, 7065–7072.

Long, K. J. (2007). *Improving contaminant source characterization and meteorological data forcing with a genetic algorithm*. Master's thesis, The Pennsylvania State University.

Long, K. J., Haupt, S. E., & Young, G. S. (2008). Improving meteorological forcing and contaminant source characterization using a genetic algorithm, to be submitted to *Optimization and Engineering*.

Loughlin, D. H., Ranjithan, S. R., Baugh, J. W., Jr., & Brill, E. D., Jr. (2000). Application of GAs for the design of ozone control strategies. *Journal of the Air & Waste Management Association*, *50*, 1050–1063.

National Research Council (2003). *Tracking and predicting the atmospheric dispersion of hazardous material releases. implications for homeland security*. Washington, DC: The National Academies Press.

Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, *7*, 308–314.

Penrose, R. (1955). A generalized inverse for matrices. *Proceedings of the Cambridge Philosophical Society*, *51*, 406–413.

Qin, R., & Oduyemi, K. (2003). Atmospheric aerosol source identification and estimates of source contributions to air pollution in Dundee, UK. *Atmospheric Environment*, *37*, 1799–1809.

Sykes, R. I., & Gabruk, R. S. (1997). A second-order closure model for the effect of averaging time on turbulent plume dispersion. *Journal of Applied Meteorology*, *36*, 1038–1045.

Sykes, R. I., Lewellen, W. S., & Parker, S. F. (1986). A gaussian plume model of atmospheric dispersion based on second-order closure. *Journal of Applied Meteorology*, *25*, 322–331.

Sykes, R. I., Parker, S. F., Henn, D. S., Cerasoli, C. P., & Santos, L. P. (1998). PC-SCIPUFF version 1.2PD technical documentation (ARAP Rep. 718). Princeton, NJ: Titan Research and Technology Division, Titan Corporation, 172 pp.

Watson, T. B., Keislar, R. E., Reese, B., George, D. H., & Biltoft, C. A. (1998). The Defense Special Weapons Agency Dipole Pride 26 field experiment (NOAA Air Resources Laboratory Tech. Memo. ERL ARL-225), 90 pp.

Wyngaard, J. C. (1992). Atmospheric turbulence. *Annual Review of Fluid Mechanics*, *24*, 205–233.

Reinforcement Learning of Optimal Controls

15

John K. Williams

15.1 Introduction

As humans, we continually interpret sensory input to try to make sense of the world around us, that is, we develop mappings from observations to a useful estimate of the "environmental state". A number of artificial intelligence methods for producing such mappings are described in this book, along with applications showing how they may be used to better understand a physical phenomenon or contribute to a decision support system. However, people don't want simply to understand the world around us. Rather, we interact with it to accomplish certain goals – for instance, to obtain food, water, warmth, shelter, status or wealth. Learning how to accurately estimate the state of our environment is intimately tied to how we then use that knowledge to manipulate it. Our actions change the environmental state and generate positive or negative feedback, which we evaluate and use to inform our future behavior in a continuing cycle of observation, action, environmental change and feedback.

In the field of machine learning, this common human experience is abstracted to that of a "learning agent" whose purpose is to discover through interacting with its environment how to act to achieve its goals. In general, no teacher is available to supply correct actions, nor will feedback always be immediate. Instead, the learner must use the sequence of experiences resulting from its actions to determine which actions to repeat and which to avoid. In doing so, it must be able to assign credit or blame to actions that may be long past, and it must balance the exploitation of knowledge previously gained with the need to explore untried, possibly superior strategies. Reinforcement learning, also called stochastic dynamic programming, is the area of machine learning devoted to solving this general learning problem. Although the term "reinforcement learning" has traditionally been used in a number of contexts, the modern field is the result of a synthesis in the 1980s of ideas from optimal control theory, animal learning, and temporal difference methods from artificial intelligence. Finding a mapping that prescribes actions based on measured environmental states in a way that optimizes some long-term measure of success is the subject of what mathematicians and engineers call "optimal control" problems and psychologists call "planning" problems. There is a deep body of mathematical literature on optimal control theory describing how to analyze a system and develop optimal mappings. However, in many applications the system is poorly understood, complex, difficult to analyze mathematically, or changing in time. In such cases, a machine learning approach that learns a good control strategy from real or simulated experience may be the only practical approach (Si et al. 2004).

This chapter begins with a brief introduction to the origins of reinforcement learning, then leads the reader through the definitions of Markov Decision Processes (MDPs), policies and value functions and the formulation of the Bellman Optimality Equation, which characterizes the solution to an MDP. The notion of Q-values is presented with a description of how they can be used to improve policies and how value functions and Q-values may be estimated from

John K. Williams (✉)
Research Applications Laboratory
National Center for Atmospheric Research
Address: P.O. Box 3000, Boulder, CO 80307, USA
Phone: 303-497-2822;
Fax: 303-497-8401;
Email: jkwillia@ucar.edu

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

an agent's experience with the environment. Optimal Q-values, which are associated with optimal policies for MDPs, may be learned through Q-learning or several related algorithms, which are described next. Partially observable MDPs, in which only partial state information is available, are discussed. It is then shown how reinforcement learning algorithms may be applied to MDPs for which the state and action spaces are large or continuous through the use of function approximation, including neural networks. Finally, three sample applications are presented: dynamic routing in a wireless sensor array, control of a scanning remote sensor, and optimal aircraft route selection given probabilistic weather forecasts. Some readers may wish to read the first few sections and then jump to the applications, returning as needed to the theoretical sections as needed to understand the notation and techniques illustrated there.

15.2 History and Background

As mentioned above, the field of reinforcement learning synthesizes ideas from the fields of mathematics, engineering, and psychology. A key mathematical concept is dynamic programming, developed in the 1950s by Richard Bellman (Bellman 1957). Bellman built on earlier work by Hamilton and Jacobi, including the formulation of the Hamilton-Jacobi equation for classical mechanics (Hamilton 1835). Dynamic programming addresses how to find optimal controls in dynamical systems subject to a degree of randomness. In a standard formulation of the problem, actions drive transitions from one state of the system, or "environment," to another, with each transition accompanied by a "cost". Both the transitions and the costs may be random variables that are functions of the starting state and the action taken. Bellman showed that the optimal control strategy in such a problem can be determined from the solution to a certain equation, now called the Bellman or Hamilton-Jacobi-Bellman Optimality Equation (see Section 15.5). Dynamic programming techniques solve the optimal control problem by solving this equation, and it is fundamental to the reinforcement learning methods described in this chapter.

On the other hand, reinforcement theories of animal learning are a cornerstone of psychology. It has long been recognized that an animal's behavior when presented with a given situation can be modified by rewarding or punishing certain actions. Trial-and-error learning algorithms for computers based on this sort of feedback or "reinforcement" date back to the work of A. M. Turing in the 1940s (Turing 1948, 1950). The idea of temporal difference learning, described in Section 15.8, may also owe its genesis to psychology via the concept of a *secondary reinforcer*. In animal behavior studies, a secondary reinforcer may be paired with a primary reinforcer (reward or punishment) through training. After establishing the association with the primary reinforcer, the animal may be trained using the secondary reinforcer *in place* of the primary reinforcer. For example, a pigeon may be trained to perform a task to receive a food reward, where the food is accompanied by a musical tone. Later, the pigeon can be taught to learn tasks for which the "reward" is the tone itself, even if a food treat no longer accompanies it. Temporal-difference methods provide a way to propagate feedback information "backward" in a sequence of experiences so that actions leading to a successful outcome are reinforced even when their ultimate payoff is significantly delayed. The successful use of temporal-difference algorithms in artificial intelligence dates back to Arthur Samuel's famous checkers-playing program (Samuel 1959), and has been used successfully for many game playing and other practical applications since.

These ideas – dynamic programming, trial-and-error learning, and temporal differences – were brought together by Chris Watkins in his 1989 Ph.D. dissertation, *Learning from Delayed Rewards*, in which he presented the "Q-learning" algorithm (Watkins 1989; see also Watkins and Dayan 1992). The simplicity and versatility of Q-learning quickly made it one of the most popular and widely used reinforcement learning algorithms, and its invention helped spawn the expansion of reinforcement learning into a broad field of research. Dozens, if not hundreds, of reinforcement learning algorithms have since been proposed and applied to a wide variety of optimal control problems.

In real-world applications, the possible environmental states and actions for an optimal control problem often cannot reasonably be enumerated, so learning a "lookup table" that prescribes an action for each possible state may not be practical. Reinforcement learning algorithms accommodate large or continuous state and action spaces by using function approximators, including linear maps and neural networks, to represent the information needed to map

states to actions. Neural networks offer the advantage that well-formulated methods exist for incrementally updating their parameters as new "training" data become available, and they often work well when states and actions are encoded in such a way that similar states and actions reliably lead to similar state transitions and feedback. Because of the popularity of this approach, some researchers have coined the term "neuro-dynamic programming" to describe the fusion of ideas from neural networks and dynamic programming (Bertsekas and Tsitsiklis 1996). However, reinforcement learning with function approximation has been shown to be unstable and to fail to converge for some problems and algorithms. The theory and methods described in this chapter are presented first for finite state and action spaces, thereby illustrating many essential issues in reinforcement learning in this more straightforward domain. The integration of function approximation with these algorithms is discussed in Section 15.12.

Of course, it is only possible to cover a few highlights of reinforcement learning theory and its applications in this single short introductory chapter. For a more comprehensive treatment, the reader is referred to the excellent texts *Neuro-Dynamic Programming* by Bertsekas and Tsitsiklis (1996) and *Reinforcement Learning* by Sutton and Barto (1998).

15.3 Markov Decision Processes

Before reinforcement learning algorithms can be described in detail, it is first necessary to define the problems to which they apply, which are called *Markov decision processes* or *Markov decision problems* (MDPs). In an MDP, the learning agent is able to interact with its environment in a limited fashion: it observes the *state* of the environment, chooses one of several available *actions*, and receives feedback, or *reinforcement*, in the form of some numerical *cost* or *reward* as a result of the action taken and the resulting state transition. The goal of the agent is to devise a *policy* – a rule for choosing actions based on observed states – that minimizes some measure of long-term costs (or, equivalently, maximizes long-term rewards). This policy could be either a deterministic mapping from states to actions or a rule specifying a probability distribution from which an action is to be randomly chosen in each state. Figure 15.1 shows a diagram of

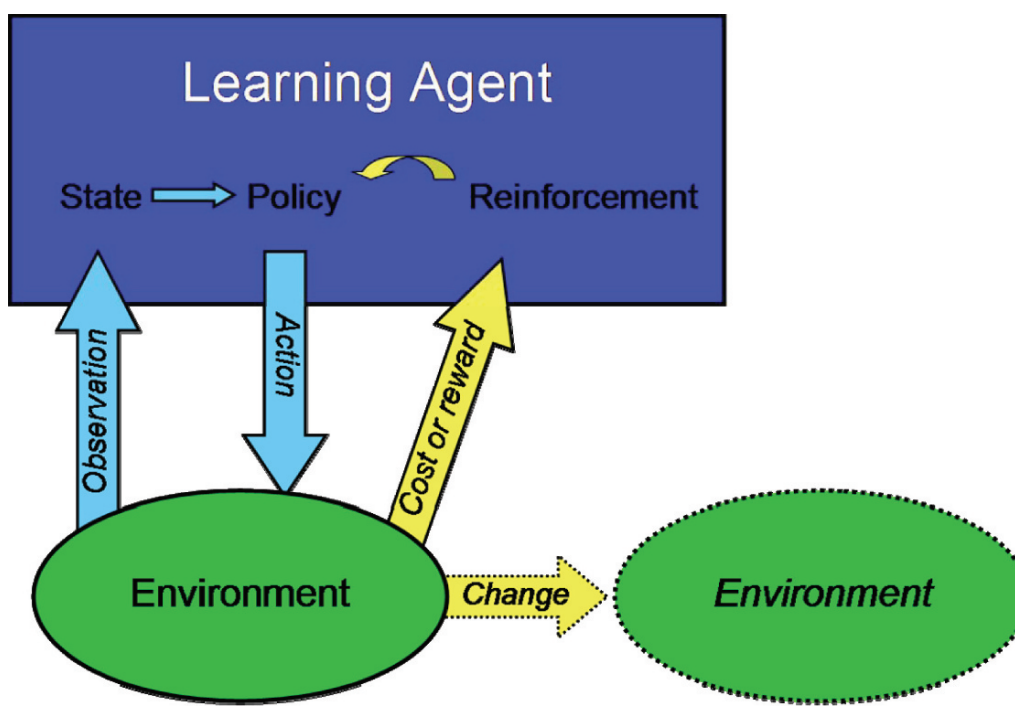


Fig. 15.1 Diagram of a reinforcement learning agent interacting with its environment: observations provide information about the environmental state that in turn may be used to determine an action based on the current policy. The action may result in a change to the environment and a cost or reward reinforcement signal that can be used to modify the policy.

this interaction between learning agent and a real or simulated environment. The key feature of an MDP (the "Markov property") is that the probabilities of state transitions and costs are a function only of the present state and action taken, not the past history of actions or states. In other words, the state of the environment by itself contains all the information needed to determine the result of a course of action, at least in a probabilistic sense.

To be more mathematically precise, the possible states obtained from observations of the environment comprise a set $\boldsymbol{S}$, and from each state $i \in \boldsymbol{S}$ there is a set of available actions that may be denoted $U(i)$. For the sake of simplicity, we shall initially assume that both $\boldsymbol{S}$ and $U(i)$ are finite sets, though infinite and even continuous state and action spaces may be dealt with via function approximation (see Section 15.12). The probability of a transition from state i to state j under action u may be written as $P^u(i, j)$ and is assumed not to change as a function of time. The cost of a state transition from i to j under action u is a random variable $g(i, u, j)$ that we will assume has a finite expected (mean) value $\bar{g}(i, u, j)$ and finite variance. As in the field of economics, where the same approach is used to compute the present value of future assets and liabilities, the MDP is equipped with a discount factor, α, a number between 0 and 1. A cost incurred one time unit in the future must be multiplied by the discount factor to compare it to an immediate cost. Thus if $\alpha = 1$, it means that future costs are valued the same as present costs; if $\alpha < 1$, future costs are less significant than immediate costs of the same amount. For example, given an inflation rate of 5% per timestep, α might be about 0.95, meaning that a future asset or liability of \$1.00 next year is worth only \$0.95 in current value, making it worthwhile to take rewards immediately, while their values are highest, while delaying the payment of fixed costs as long as possible. MDPs may be categorized according to the value of α. If $\alpha < 1$, the MDP is called a *discounted problem* (DCP), since future costs are discounted. If $\alpha = 1$ and there is a "final" state that ends the process (usually denoted as state 0), it is called a *stochastic shortest path problem* (SSPP) since a typical problem of this sort is finding a shortest path through a maze or a network. For a thorough exposition of MDPs, the reader is referred to the text by Puterman (2005). A summary of the symbols used in this chapter for describing MDPs and the concepts and reinforcement learning algorithms for solving them may be found in Table 15.1.

If the transition probability values $P^u(i, j)$ and average costs $\bar{g}(i, u, j)$ are known to the learning

Table 15.1 Descriptions of some common symbols used in this chapter to describe MDPs and reinforcement learning algorithms.

Symbol	Description
$\boldsymbol{S}$	Set of environmental states for an MDP.
$U(i)$	Set of actions available from state i.
$P^u(i, j)$	Probability of transition from state i to state j under action u.
$g(i, u, j)$	Random variable cost of transition from state i to state j under action u.
$\bar{g}(i, u, j)$	Expected (mean) cost of transition from state i to state j under action u.
α	Discount factor between 0 and 1.
i_t, u_t	State and action taken at time t.
μ	A deterministic policy, which maps states to actions.
π	A stochastic policy, which maps state-action pairs to probabilities.
J^μ	The value function for the policy μ; see (15.1).
J^*	The optimal value function.
Q^μ	The action-value function, or Q-value, for the policy μ; see (15.13).
Q^*	The optimal Q-value.
J_t, Q_t	Value function and Q-value iterates for a learning algorithm at time t.
γ	Learning rate or step size parameter.
$R_t^{(N)}$	N-step return beginning at time t; see (15.22).
R_t^λ	λ–return (average of N-step returns) beginning at time t; see (15.23).
δ_t	One-step temporal difference at time t; see (15.25).
e	Eligibility trace for a state or state-action pair.
$\boldsymbol{M}$	Set of messages for a POMDP.

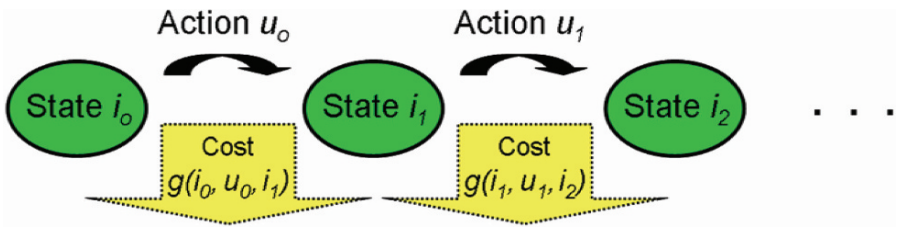


Fig. 15.2 A trajectory of an MDP, with a sequence of actions generating state transitions accompanied by feedback signals ("costs").

agent for all states i, j and actions u, we say that a *model* of the environment is available. When a model is not provided, the learning problem is *model-free*. Some reinforcement learning algorithms use experience (e.g., simulation with various choices of actions) to estimate a model of the environment, then use dynamic programming methods to solve for the optimal value function and policy. Other algorithms, such as Q-learning and temporal difference learning, are model-free, operating directly on the cost and transition experiences as they are obtained. To refer to the sequence of states, actions, and costs experienced by a learning agent in these algorithms, we use subscripts to count the timesteps. So, for instance, i_t refers to the state occupied at the t^{th} timestep, u_t refers to the t^{th} action, and the cost incurred in the transition to the next state is $g(i_t, u_t, i_{t+1})$, which is sampled from a random variable having mean value $\bar{g}(i_t, u_t, i_{t+1})$ and finite variance. The trajectory of the MDP is then written as the sequence of states, actions, and costs: i_0, u_0, $g(i_0, u_0, i_1)$, i_1, u_1, $g(i_1, u_1, i_2), \ldots$ (see Fig. 15.2). The goal of a reinforcement learning algorithm is to use such a sequence of experiences to determine an optimal policy.

15.4 Policies and Value Functions

Solving an MDP means finding an optimal policy: a mapping from states to actions which, when used for action selection, minimizes some measure of long-term costs. For an MDP, it turns out that there is always an optimal policy that minimizes the expected sum of future discounted costs for each state (see below); this policy is deterministic and does not change as a function of time. However, in the process of finding an optimal policy, a learning agent must balance between acting according to the best policy found so far and exploring new, untried actions that may lead to even better results. This is the famous "exploitation vs. exploration" conundrum of reinforcement learning. One way to perform this balance is to use action selection that includes some degree of randomness while learning, that is, a *stochastic* policy that specifies a probability distribution over available actions for each state. Defining these concepts formally, a *deterministic policy* μ is a mapping from states to actions such that $\mu(i) \in U(i)$ for all states $i \in \boldsymbol{S}$. A *stochastic policy* π is a mapping from state-action pairs to probabilities such that (a) $0 \leq \pi(i, u) \leq 1$ for all states $i \in \boldsymbol{S}$ and actions $u \in U(i)$, (b) $\pi(i, u) = 0$ whenever $u \notin U(i)$, and (c) $\sum_{u \in U(i)} \pi(i, u) = 1$ for each state $i \in \boldsymbol{S}$. The sequence of states encountered in the MDP while acting under a fixed policy is called a *Markov chain*. For SSPPs, the policies that create a Markov chain that is guaranteed to terminate (with probability one) are called *proper* policies.

In order to evaluate a policy, we must specify an objective way to measure its success. A common metric, and the main one used in this chapter, is the "expected" (i.e., mean) total long-term discounted cost incurred from executing the policy. This long-term expected cost is a function of the starting state, and is called the policy's *value function*. Formally, the value function for the deterministic policy μ, written J^{μ}, is defined for each state $i \in \boldsymbol{S}$ by

$$J^{\mu}(i) = E\left[\sum_{t=0}^{\infty} \alpha^t g(i_t, \mu(i_t), i_{t+1}) \mid i_0 = i\right] \quad (15.1)$$

The "E" in (15.1) denotes the expected value – the mean or average over all the randomness in the state transitions and costs g resulting from the prescribed actions – and the vertical line "|" means "such that". Thus, this equation defines the value function for state i as the expected value of the infinite sum of future

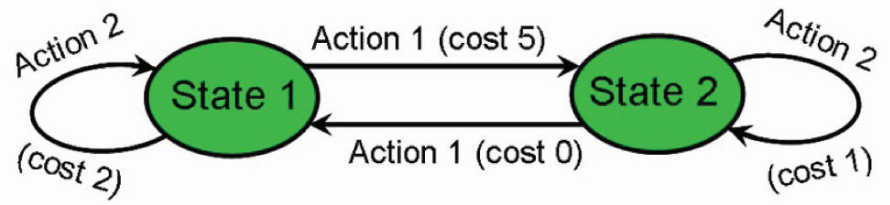


Fig. 15.3 Example MDP with two states, two available actions and deterministic state transitions represented by the arrows. Costs associated with the state transitions are indicated in parentheses, and $\alpha = 0.9$, making this a discounted problem (DCP).

discounted costs incurred when beginning in state i and choosing actions according to the policy μ. The value function for a stochastic policy π is denoted J^π and is similarly defined by

$$J^\pi(i) = E\Bigg[\sum_{t=0}^{\infty} \alpha^t g(i_t, u_t, i_{t+1}) \mid i_0 = i\,; \quad \text{each } u_t \text{ chosen according to } \pi\Bigg] \tag{15.2}$$

We will make assumptions on the MDPs and policies that ensure that the value functions defined in (15.1) and (15.2) are finite; for instance, this will be true in the common case that the costs g are bounded and the discount factor $\alpha < 1$.

Rewriting the expectation in (15.1) by pulling out the first term of the sum leads to the so-called *consistency* or *Bellman* equation:

$$\begin{aligned} J^\mu(i) &= E_j\left[g(i, \mu(i), j) + \alpha J^\mu(j)\right] \\ &= \sum_{j \in S} P^{\mu(i)}(i, j)\ (\bar{g}(i, \mu(i), j) + \alpha J^\mu(j)) \end{aligned} \tag{15.3}$$

where the expectation E_j is over the (possibly) random cost g and random transition to a next state j. It can be shown that, for any deterministic policy, μ, J^μ is the *unique* solution to the consistency equation, that is (15.3) completely characterizes J^μ (this is a corollary to the Bellman Optimality Equation presented in the next section). The consistency equation for stochastic policies may be written analogously.

An *optimal value function* is one that has the smallest achievable value for each state. That is, if we denote the optimal value function by J^*, then for all states $i \in \boldsymbol{S}$, $J^*(i) \leq J^\pi(i)$ for all policies π and $J^*(i) = J^\pi(i)$ for some policy π. It will be shown in the next section that, under certain reasonable assumptions, there is at least one deterministic policy μ such that $J^*(i) = J^\mu(i)$ for all states $i \in \boldsymbol{S}$. Briefly, this is a result of the fact that, due to (15.3), any change of a policy that reduces the expected discounted sum of future costs from one state can only cause an improvement in other states as well. Naturally, any policy whose value function is equal to the optimal value function is called an *optimal policy*. The next section characterizes such policies. First, though, we consider two examples to make these concepts more concrete. Figure 15.3 shows a representation of a two-state MDP with discount factor $\alpha = 0.9$, making it a DCP. In State 1, two actions are possible: Action 1 causes a transition to State 2 with a cost of 5, and Action 2 exacts a cost of 2 but doesn't change the state. In State 2, Action 1 causes a transition to State 1 with a cost of 0, and Action 2 loops back to State 2 with a cost of 2. The four possible policies are given in Table 15.2 along with their value functions. For example, the policy here arbitrarily called Policy 2, given by $\{\mu(1) = 1, \mu(2) = 2\}$, has transition probabilities $P^{\mu(1)}(1, 1) = 0$, $P^{\mu(1)}(1, 2) = 1$, $P^{\mu(2)}(2, 1) = 0$ and $P^{\mu(2)}(2, 2) = 1$ with costs $\bar{g}(1, \mu(1), 2) = 5$ and $\bar{g}(2, \mu(2), 2) = 1$. Under this policy, if the initial state is State 1, the first step will transition to State 2 with

Table 15.2 Policies and value functions for the MDP depicted in Fig. 15.3

Policy 1	$\{\mu(1) = 1, \mu(2) = 1\}$	$J(1) = 500/19 \approx 26.3$ $J(2) = 450/19 \approx 23.7$
Policy 2	$\{\mu(1) = 1, \mu(2) = 2\}$	$J(1) = 14$ $J(2) = 10$
Policy 3	$\{\mu(1) = 2, \mu(2) = 1\}$	$J(1) = 20$ $J(2) = 18$
Policy 4	$\{\mu(1) = 2, \mu(2) = 2\}$	$J(1) = 20$ $J(2) = 10$

a cost of 5 and then the Markov chain will remain in State 2 with an additional cost of 1 per timestep. Thus, the value function for this policy may be computed via (15.1) as

$$\begin{aligned} J^{\mu}(1) &= 5 + \alpha \cdot 1 + \alpha^2 \cdot 1 + \alpha^3 \cdot 1 + \ldots \\ &= 5 + \sum_{t=1}^{\infty} \alpha^t = 5 + \frac{\alpha}{1-\alpha} = 5 + 9 = 14 \\ J^{\mu}(2) &= 1 + \alpha \cdot 1 + \alpha^2 \cdot 1 + \alpha^3 \cdot 1 + \ldots \\ &= \sum_{t=0}^{\infty} \alpha^t = \frac{1}{1-\alpha} = 10 \end{aligned} \tag{15.4}$$

or by solving the system of equations given by (15.3):

$$\begin{aligned} J^{\mu}(1) &= 5 + \alpha J^{\mu}(2) \\ J^{\mu}(2) &= 1 + \alpha J^{\mu}(2) \end{aligned} \tag{15.5}$$

which yields the same result. If the transitions had a random element so that, for instance, Action 2 in State 2 would occasionally cause a transition to State 1, then (15.1) or (15.3) would become a bit more complicated, accounting for the cost of all possible trajectories along with their probabilities, but the main idea remains the same. As can be seen from Table 15.2, Policy 2 is the optimal policy since its value function is smallest.

A second problem, a very small "robot maze" SSPP, is depicted in Fig. 15.4. In both State 1 and State 2, the robot may attempt to move north, south, east or west, but the only possible transitions to a new state are from State 1 to State 2 by moving east or from State 2 to State 0 by moving south. Each attempted move incurs a cost of 1, representing the robot's energy usage or the elapsed time, and the trajectory ends when State 0 is reached, representing escape from the maze. The optimal policy is the one that leads most quickly out of the maze, which in this case is $\{\mu(1) = \text{East}, \mu(2) = \text{South}\}$. For this policy, $J^{\mu}(1) = 2$ and $J^{\mu}(2) = 1$; for any other deterministic policy μ, $J^{\mu}(1) = \infty$, and $J^{\mu}(2) = \infty$ unless $\mu(2) = \text{South}$, in which case $J^{\mu}(2) = 1$. Thus $\{\mu(1) = \text{East}, \mu(2) = \text{South}\}$ is the optimal policy, and in fact is the only proper deterministic policy because it is the only one that produces a trajectory guaranteed to terminate regardless of the initial state. Again, these calculations would become more complex if the transitions were not deterministic, e.g., if there were some chance that the robot could knock over a wall or trip when passing through a doorway.

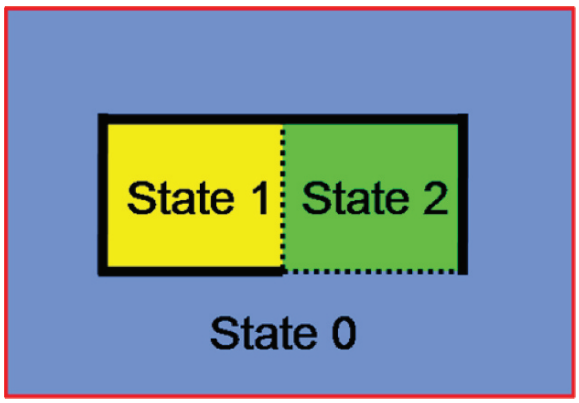


Fig. 15.4 Three state "robot maze" problem. Dotted lines represent open doorways, and solid lines represent walls. The robot may attempt to move north, south, east or west, but will only be successful in moving east from State 1 and west or south from State 2. Each move has a cost of 1, and the discount factor $\alpha = 1$, making this a stochastic shortest path problem (SSPP).

15.5 The Bellman Optimality Equation

In the 1950s, Richard Bellman formulated an equation – or, more precisely, a system of equations – that characterizes the optimal value function for an MDP (Bellman 1957). This equation forms the cornerstone of the classical theory of dynamic programming, and it is essential to reinforcement learning theory as well. The following version is adapted from those stated and proved in Bertsekas (1995) for SSPPs and Bertsekas and Tsitsiklis (1996) for DCPs.

The Bellman Optimality Equation

Suppose that an MDP is either a DCP or an SSPP such that there exists a proper deterministic policy and such that, for every improper deterministic policy, μ, $J^{\mu}(i) = \infty$ for some $i \in \boldsymbol{S}$. Then the optimal value function J^* exists, has finite components, and is the unique solution to the equations

$$\begin{aligned} J^*(i) &= \min_{u \in U(i)} E_j \left[g(i,u,j) + \alpha J^*(j) \right] \\ &= \min_{u \in U(i)} \sum_{j \in S} P^u(i,j) (\bar{g}(i,u,j) \\ &\quad + \alpha J^*(j)) \end{aligned} \tag{15.6}$$

for all states $i \in \boldsymbol{S}$, where the expectation is over the random cost g and next state j.

All theoretical results presented in this chapter for MDPs will assume that the MDP under discussion satisfies the hypotheses of the Bellman Optimality Equation.

Returning to the examples from the previous section, we note that for the DCP the optimal value function with $J^*(1) = 14$ and $J^*(2) = 10$ does satisfy (15.6) because

$$\begin{aligned}
&\min_{u\in U(i)} E_j\left[g(1,u,j)+\alpha J^*(j)\right] \\
&\quad = \min\left\{\bar{g}(1,1,2)+\alpha J^*(2), \bar{g}(1,2,1)+\alpha J^*(1)\right\} \\
&\quad = \min\{5+(0.9)(10),\ 2+(.9)(14)\} \\
&\quad = \min\{14,\ 14.6\} = 14 = J^*(1) \qquad (15.7)
\end{aligned}$$

and

$$\begin{aligned}
&\min_{u\in U(i)} E_j\left[g(2,u,j)+\alpha J^*(j)\right] \\
&\quad = \min\left\{\bar{g}(2,1,1)+\alpha J^*(1), \bar{g}(2,2,2)+\alpha J^*(2)\right\} \\
&\quad = \min\{0+(0.9)(14),\ 1+(.9)(10)\} \\
&\quad = \min\{12.6,\ 10\} = 10 = J^*(2) \qquad (15.8)
\end{aligned}$$

Similarly, for the SSPP example,

$$\begin{aligned}
&\min_{u\in U(i)} E_j\left[g(1,u,j)+\alpha J^*(j)\right] \\
&\quad = \min\left\{\bar{g}(1,\text{“east”},2)+\alpha J^*(2),\ \bar{g}(1,\text{“south”},1)\right. \\
&\qquad \left.+\alpha J^*(1),\ \bar{g}(1,\text{“west”},1)+\alpha J^*(1),\ldots\right\} \\
&\quad = \min\{1+(1)(1),\ 1+(1)(\infty),\ 1+(1)(\infty), \\
&\qquad 1+(1)(\infty)\} = 2 = J^*(1) \qquad (15.9)
\end{aligned}$$

and

$$\begin{aligned}
&\min_{u\in U(i)} E_j\left[g(2,u,j)+\alpha J^*(j)\right] \\
&\quad = \min\left\{\bar{g}(2,\text{East},2)+\alpha J^*(2), \bar{g}(2,\text{South},0)\right. \\
&\qquad \left.+\alpha J^*(0), \bar{g}(2,\text{West},1)+\alpha J^*(1),\ \ldots\right\} \\
&\quad = \min\{1+(1)(1),\ 1+(1)(0), 1+(1)(2), \\
&\qquad 1+(1)(1)\} = 1 = J^*(2) \qquad (15.10)
\end{aligned}$$

The Bellman Optimality Equation is important for several reasons. First, its characterization of the optimal value function is useful for theoretical purposes. Second, it leads to direct methods for computing the optimal value function. One procedure, called *value iteration*, performs an iteration based on (15.6) to successively approximate J^*. The Bellman Optimality Equation can also be written as a linear program, which can be solved by classical methods for small state and action spaces (Bertsekas and Tsitsiklis 1996, p. 37):

$$\begin{aligned}
&\text{maximize} \sum_{i\in S} J(i) \\
&\text{subject to } J(i) \le \sum_{j\in S} P^u(i,j)\ \left(\bar{g}(i,u,j)+\alpha J(j)\right) \qquad (15.11)
\end{aligned}$$

Third, the Bellman Optimality Equation provides a connection between the optimal value function and optimal policies. It can be shown that a deterministic policy μ is optimal *if and only if*

$$\begin{aligned}
&E_j\left[g(i,\mu(i),j)+\alpha J^*(j)\right] \\
&\quad = \min_{u\in U(i)} E_j\left[g(i,u,j)+\alpha J^*(j)\right] \qquad (15.12)
\end{aligned}$$

for all states $i \in \boldsymbol{S}$ (see Williams 2000). That is, a policy is optimal if and only if the action it prescribes for each state is one that achieves the optimal value function as characterized by the Bellman Optimality Equation.

If a model of the MDP is available, the transition probabilities $P^u(i,j)$ and expected costs $\bar{g}(i,u,j)$ may be substituted into (15.6) and it can be solved iteratively, or the linear program in (15.11) could be used. Once J^* is found, (15.12) determines an optimal policy. If a model is not available, a learning agent could use exploratory actions and their resulting costs and state transitions to create one by estimating the transition probabilities and associated expected costs, then use one of these approaches. Alternatively, the learning agent could use the same experience to iteratively improve an estimate of the optimal value function, but then an alternative to (15.12) must be found for determining an optimal policy. This approach to learning from experience can be achieved via iterative methods that use an extended concept of value functions that apply to pairs of states and actions, as described in the following section.

15.6 *Q*-Values

The total expected future discounted cost obtained when a certain action is executed in some state and then a fixed policy is followed thereafter is called an *action value function* or, for historical reasons, a *Q*-factor or *Q*-value. *Q*-values are particularly useful in determining how to change a policy to improve

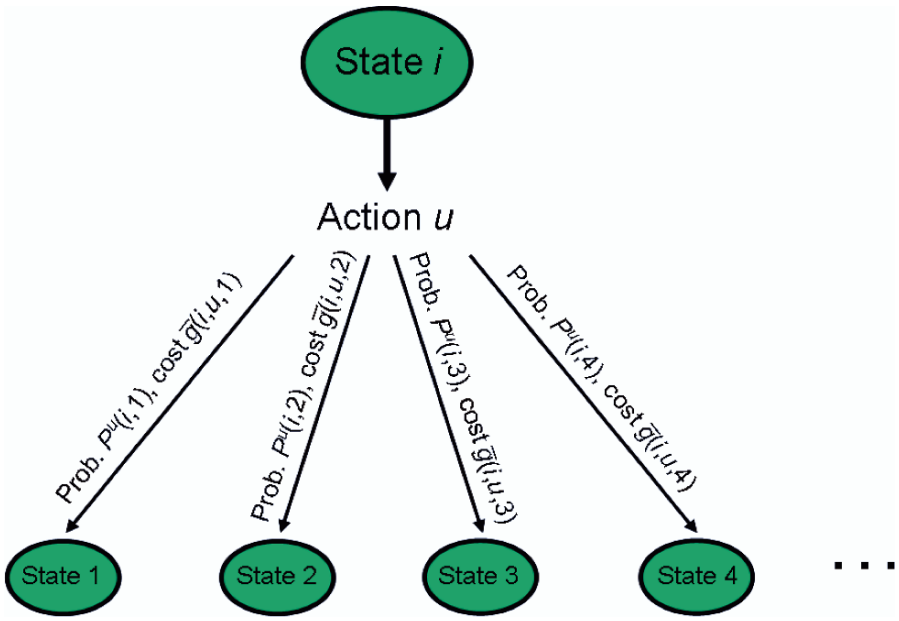


Fig. 15.5 Diagram of the possible transitions from taking action u in state i, along with their probabilities and expected costs. The Q-value for a policy μ, Q^μ, may be computed by taking a probability-weighted combination of all the transition costs plus discounted value functions of the resulting states, J^μ, as given by (15.14). The optimal Q-value, Q^*, is similarly computed from the optimal value function, J^*, in (15.15).

it. Furthermore, the optimal policy for an MDP can be determined directly from its optimal Q-value, denoted Q^*, without requiring a model of the environment. Formally, we define the Q-value for a policy μ by

$$Q^\mu(i,u) = E\left[g(i_0,u,i_1) + \sum_{t=1}^{\infty} \alpha^t g(i_t,\mu(i_t),i_{t+1}) \;\; | \, i_0 = i \right] \tag{15.13}$$

Here the expectation is taken over all possible trajectories and transition costs under the policy μ after action u is taken in state i. Note that (15.13) is different from (15.1) only in that the action taken in state i_0 is not necessarily the one prescribed by the policy μ; however, $Q^\mu(i,\ \mu(i)) = J^\mu(i)$. The Q-value for a stochastic policy π is defined similarly to (15.13), but with all actions following the first one being chosen according to π.

As is true for the value function, there is a consistency equation for Q-values that follows immediately from their definition and the definition of the value function. For any deterministic policy μ,

$$\begin{aligned} Q^\mu(i,u) &= E_j\left[g(i,u,j) + \alpha J^\mu(j) \right] \\ &= \sum_{j \in S} P^u(i,j) \;\; (\bar{g}(i,u,j) + \alpha J^\mu(j)) \end{aligned} \tag{15.14}$$

for each state $i \in \boldsymbol{S}$ and action $u \in U(i)$, where the expectation is over the random cost g and next state j. A similar equation holds for stochastic policies. The right side of (15.14) may be thought of as following all the possible state transitions and costs that can result from taking action u in state i and weighting the results – the transition costs plus the discounted value function of the resultant state – according to their probabilities. This is diagrammed in Fig. 15.5. Equation (15.13) could also be represented by an infinitely large tree of the possible trajectories, where the discounted costs of each transition weighted by their probabilities are added up over all the branches.

An *optimal Q-value* is one which has the smallest achievable value for each state-action pair. That is, if we denote the optimal Q-value function by Q^*, then for all states $i \in \boldsymbol{S}$ and actions $u \in U(i)$,

$Q^*(i,u) \le Q^\pi(i,u)$ for all (deterministic or stochastic) policies π and $Q^*(i,u) = Q^\pi(i,u)$ for some policy π. It follows from (15.14) and the definition of the optimal value function that the optimal Q-value satisfies the equation

$$Q^*(i,u) = E_j\left[g(i,u,j) + \alpha J^*(j)\right] \qquad (15.15)$$

for all states $i \in \boldsymbol{S}$ and actions $u \in U(i)$. Thus (15.6) may be written in terms of the optimal Q-value as

$$J^*(i) = \min_{u \in U(i)} Q^*(i,u) \qquad (15.16)$$

The Bellman Optimality Equation for value functions can also be stated in terms of Q-values. In particular, under the hypotheses for the Bellman Optimality Equation, the optimal value function Q^* exists, has finite components, and is the unique solution to the equations

$$\begin{aligned} Q^*(i,u) &= E_j\left[g(i,u,j) + \alpha \min_{v \in U(j)} Q^*(j,v)\right] \\ &= \sum_{j \in S} P^u(i,j)\left(\bar{g}(i,u,j)\right. \\ &\quad \left. + \alpha \min_{v \in U(j)} Q^*(j,v)\right) \end{aligned} \qquad (15.17)$$

for all states $i \in \boldsymbol{S}$ and actions $u \in U(i)$. Furthermore, it follows from (15.12) and (15.15) that the deterministic policy μ is optimal if and only if

$$Q^*(i,\mu(i)) = \min_{u \in U(i)} Q^*(i,u) \qquad (15.18)$$

for all states $i \in \boldsymbol{S}$. According to this characterization, an optimal policy can be determined directly from the optimal Q-value even in the absence of a model of the environment by simply selecting an action in each state that has the smallest value of Q^*. This is a powerful advantage for a wide range of problems.

In words, (15.15) or (15.17) define $Q^*(i,u)$ as the total of the future discounted costs expected from taking action u in state i, assuming that the optimal policy is followed for every action thereafter. The "robot maze" problem described earlier and illustrated in Fig. 15.4 allows a particularly concrete interpretation: $Q^*(i,u)$ is the expected minimum number of steps from state i to the exit (state 0), including the result of action u. The values of Q^* for this problem are depicted in Fig. 15.6. For example, $Q^*(2, \text{East}) = 2$ because trying to move east from State 2 (and bouncing off the wall) uses one step, and then moving south and out of the maze requires a second step. The shortest path to the exit can be found by simply moving in the direction associated with the smallest value of Q^* in each state. Thus, finding the optimal Q-value is tantamount to finding the optimal policy.

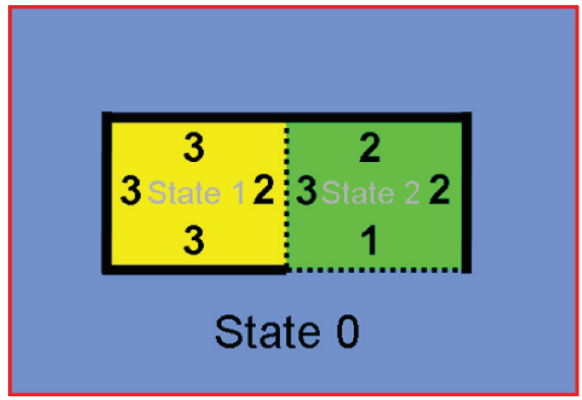


Fig. 15.6 The three-state robot maze with optimal Q-values displayed for each state in the direction of the associated move. For this problem, the Q^* values represent the minimum number of steps to the "exit" including the given action.

15.7 Policy Improvement

Even when a model of the environment is available, there are various reasons to seek other solution methods for an MDP than value iteration using (15.6) or the linear program in (15.11). Value iteration can be computationally intensive, and other methods may often be faster. Linear programming techniques are effective for small state and action spaces, but may not work well for large problems. And a model of the MDP that is estimated from experience may have errors that lead to a poor solution with either approach. Finally, if something is known *a priori* about a good policy, a learning framework that can make use of that knowledge is desirable. For these reasons, a wide variety of reinforcement learning algorithms that make direct use of experience obtained from interacting with the environment have been proposed. Many of these use some sort of *policy iteration*. At the root of this method is the observation that, from (15.14) and the fact that $J^\mu(i) = Q^\mu(i,\mu(i))$ it follows that for any two policies μ and μ', if $Q^\mu(i,\mu'(i)) \le Q^\mu(i,\mu(i))$ for all states $i \in \boldsymbol{S}$, then $J^{\mu'}(i) \le J^\mu(i)$ for all $i \in \boldsymbol{S}$; if in addition $Q^\mu(i,\mu'(i)) < Q^\mu(i,\mu(i))$ for some state $i \in \boldsymbol{S}$, then $J^{\mu'}(i) < J^\mu(i)$ for that state. This result is called the Policy Improvement Theorem. Policy

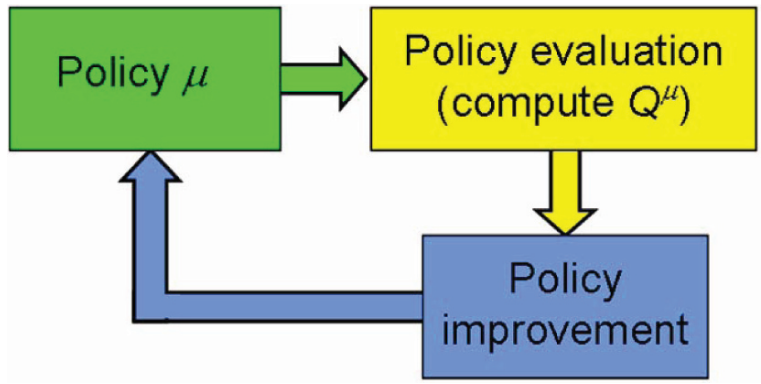


Fig. 15.7 A high-level diagram illustrating policy iteration: the cycle of improvement may continue until an optimal policy is found. Many practical reinforcement learning algorithms have this form.

iteration operates by iteratively improving a policy until it becomes optimal. This is done by evaluating the Q-value for a policy and then choosing a new policy that, for each state, uses an action that appears better (or at least equally good) based on the Q-value. It may be described more precisely as follows.

Policy Iteration Algorithm

Select an initial policy μ_0, either at random or based on *a priori* knowledge of a good solution to the MDP. At each subsequent time $t \geq 0$, evaluate Q^{μ_t} exactly and update the policy according to

$$\mu_{t+1}(i) = \begin{cases} \mu_t(i) \text{ if } Q^{\mu_t}(i, \mu_t(i)) \\ \quad = \min_{u \in U(i)} Q^{\mu_t}(i, u), \text{ or} \\ \text{any element } v \text{ for which } Q^{\mu_t}(i, v) \\ \quad = \min_{u \in U(i)} Q^{\mu_t}(i, u) \text{ otherwise} \end{cases} \tag{15.19}$$

for all states $i \in \boldsymbol{S}$. Repeat this process until $\mu_{t+1} = \mu_t$, then stop. μ_{t+1} is an optimal policy.

A high-level diagram of the Policy Iteration Algorithm is shown in Fig. 15.7. The Policy Iteration Algorithm will eventually end because there are only a finite number of possible policies, and at each step the policy improvement theorem guarantees that $J^{\mu'}(i) \leq J^{\mu}(i)$ for all $i \in \boldsymbol{S}$ and $J^{\mu'}(i) < J^{\mu}(i)$ for at least one state $i \in \boldsymbol{S}$, preventing any policy from being repeated.

Policy iteration provides a method for improving a policy once its Q-values for all state-action pairs are known, but it does not address how these values are to be obtained. The process of computing the value function or Q-value for a policy is called *policy evaluation* or *prediction*. If a model of the environment is known, computing the value function is sufficient since each Q-value component can be obtained from a "one-step lookahead" using (15.14). In this case, there are a number of classical methods in dynamic programming for policy evaluation, including the iterative solution of (15.3). Another possibility is to use a Monte-Carlo approach: execute the policy many times starting from each state and average the returns, as suggested by (15.1); the law of large numbers ensures that this average eventually converges to the value function. The temporal difference algorithms discussed in the next section provide a parameterized combination of these two approaches that may be used in the model-free case.

Algorithms that use the framework of policy iteration but do not evaluate the Q-value exactly before updating the policy are commonly called *generalized* or *optimistic* policy iteration algorithms. Many practical reinforcement learning algorithms are of this type.

15.8 Temporal Difference Learning

Temporal difference (TD) algorithms provide a way for a learning agent to learn from experience with the environment. The basic idea is to update an estimate of the MDP's value function based on the difference of successive estimates of the value of a state as new experience is obtained. For instance, suppose an estimate at time t of the value of state i_t for the policy μ is $J_t(i)$. Starting in state i_t and executing the action

$\mu(i_t)$ may result in a transition to state j accompanied by a cost $g(i_t, \mu(i_t), j)$, and the new state j will have an associated value function estimate $J_t(j)$. The quantity $g(i_t, \mu(i_t), j) + \alpha J_t(j)$ is an estimate of the right-hand side of (15.3), called the *one-step return*. Of course, this estimate involves only one sample of what may be a noisy random process, and it is also impacted by whether or not $J_t(j)$ is a good estimate of $J^\mu(j)$. Therefore, it is not advisable to immediately replace $J_t(i_t)$ with this new estimate, but rather to "nudge" $J_t(i_t)$ a small step toward it. This may be accomplished by choosing a small step-size γ_t between 0 and 1 and making the assignment

$$\begin{aligned} J_{t+1}(i_t) &= (1-\gamma_t) J_t(i_t) \\ &\quad + \gamma_t \left(g(i_t, \mu(i_t), j) + \alpha J_t(j)\right) \\ &= J_t(i_t) + \gamma_t (g(i_t, \mu(i_t), j) \\ &\quad + \alpha J_t(j) - J_t(i_t)) \end{aligned} \tag{15.20}$$

This equation represents moving $J_t(i_t)$ a fraction γ_t of the distance toward $g(i_t, \mu(i_t), j) + \alpha J_t(j)$. The quantity $g(i_t, \mu(i_t), j) + \alpha J_t(j) - J_t(i_t)$ is called a *temporal difference*, and the update method just described is called a *one-step temporal difference* algorithm, or TD(0). If the iteration (15.20) is performed repeatedly for all possible initial states $i_t \in \boldsymbol{S}$ and the values of γ_t converge to 0 in an appropriate fashion, the estimates J_t will converge to J^μ. This iteration of successive "nudges" is called a *stochastic approximation* or *Robbins-Monro* approach to the solution of the consistency equation (15.3); see Robbins and Monro (1951) and Kushner and Yin (1997). A familiar example of stochastic approximation is the incremental calculation of a population mean as new sample values are obtained. For instance, suppose $M_t = \text{mean}(\{x_1, x_2, \ldots, x_t\})$ and now a new sample x_{t+1} has become available. Then the new mean, M_{t+1}, may be written

$$\begin{aligned} M_{t+1} &= \frac{1}{t+1} \sum_{n=1}^{t+1} x_n = \frac{1}{t+1} \left(t\, M_t + x_{t+1}\right) \\ &= M_t + \frac{1}{t+1} \left(x_{t+1} - M_t\right) \end{aligned} \tag{15.21}$$

This equation has the same form as (15.20), with the sample value x_{t+1} representing a new (though noisy) estimate for the population mean and the step-size $\gamma_t = (t+1)^{-1}$. Of course, as $t \to \infty$, the law of large numbers guarantees that the values of M_{t+1} defined by (15.21) will converge to the mean of the distribution from which the x_t are drawn (assuming the distribution is well-behaved, e.g., bounded). In fact, convergence to the mean will occur for any sequence γ_t between 0 and 1 so long as $\sum_{t=0}^{\infty} \gamma_t = \infty$ and $\sum_{t=0}^{\infty} \gamma_t^2 < \infty$. (Note that the second condition implies that $\gamma_t \to 0$.) These are standard step-size conditions for stochastic approximation methods. For many more results and applications of stochastic approximation, see the text by Kushner and Yin (1997).

Analogs of (15.20) can be formulated using N-step returns, that is, results from a sequence of N successive actions instead of just one, along with the value of the final state. Formally, an *N-step return* beginning at time t in state i_t is defined by

$$R_t^{(N)} = \sum_{k=0}^{N-1} \alpha^k g(i_{t+k}, u_{t+k}, i_{t+k+1}) + \alpha^N J_{t+N}(i_{t+N}) \tag{15.22}$$

As is true for one-step returns, the N-step return provides a new estimate for the value function of the initial state. Larger values of N generally will produce higher variance N-step returns since they involve more random transitions, but lower bias since the value function of the final state has less influence. In fact, if N is allowed to go to infinity, the N-step return approaches an unbiased Monte-Carlo return. An algorithm having the form of (15.20) but using N-step returns under a fixed policy might be called an N-step temporal difference algorithm. However, such algorithms are rarely used in practice. Much more common is to use a weighted average of *all* the N-step returns, parameterized by a value λ between 0 and 1. The λ-return beginning at time t in state i_t is defined by

$$R_t^\lambda = (1-\lambda) \sum_{N=1}^{\infty} \lambda^{N-1} R_t^{(N)} \tag{15.23}$$

Recall that $(1-\lambda)\sum_{N=1}^{\infty} \lambda^{N-1} = 1$ for $0 \le \lambda < 1$, so (15.23) represents a weighted average of the $R_t^{(N)}$ terms with coefficients that sum to 1. Of course, the λ-return for a state i_t cannot generally be computed at the original time the state is visited, since future experience is required to determine the N-step returns. Fortunately, the λ-return update can be decomposed in a clever way that makes it possible to continue nudging the value function towards the λ-return as that future

experience is obtained. For λ between 0 and 1, we may write the λ-return temporal difference as shown below. (Here the time subscripts on the value function estimates J have been omitted for conciseness.)

$$
\begin{aligned}
R_t^\lambda - J(i_t) &= (1-\lambda)\sum_{N=1}^{\infty}\lambda^{N-1}R_t^{(N)} - J(i_t) \\
&= (1-\lambda)\sum_{N=1}^{\infty}\lambda^{N-1}\left(\sum_{k=0}^{N-1}\alpha^k g(i_{t+k},u_{t+k},i_{t+k+1}) + \alpha^N J(i_{t+N})\right) - J(i_t) \\
&= (1-\lambda)\sum_{N=1}^{\infty}\sum_{k=0}^{N-1}\lambda^{N-1}\alpha^k g(i_{t+k},u_{t+k},i_{t+k+1}) + (1-\lambda)\sum_{N=1}^{\infty}\lambda^{N-1}\alpha^N J(i_{t+N}) - J(i_t) \\
&= (1-\lambda)\sum_{k=0}^{\infty}\sum_{N=k+1}^{\infty}\lambda^{N-1}\alpha^k g(i_{t+k},u_{t+k},i_{t+k+1}) + (1-\lambda)\sum_{N=1}^{\infty}\lambda^{N-1}\alpha^N J(i_{t+N}) - J(i_t) \\
&= \sum_{k=0}^{\infty}\lambda^k\alpha^k g(i_{t+k},u_{t+k},i_{t+k+1}) + \sum_{k=0}^{\infty}\lambda^k\alpha^{k+1}J(i_{t+k+1}) - \sum_{k=0}^{\infty}\lambda^k\alpha^k J(i_{t+k}) \\
&= \sum_{k=0}^{\infty}\lambda^k\alpha^k\left(g(i_{t+k},u_{t+k},i_{t+k+1}) + \alpha J(i_{t+k+1}) - J(i_{t+k})\right)
\end{aligned}
\tag{15.24}
$$

Thus, the difference between the λ-return and the initial value function estimate for state i_t may be written as a weighted sum of quantities very close to one-step temporal differences, except that they are based on the original estimate J instead of the estimate available when the action is taken. Denoting the one-step temporal difference at time t by

$$\delta_t = g(i_t,u_t,i_{t+1}) + \alpha J_t(i_{t+1}) - J_t(i_t) \tag{15.25}$$

the formula in (15.24) can be approximated as

$$R_t^\lambda - J_t(i_t) \approx \sum_{k=0}^{\infty}\lambda^k\alpha^k\delta_{t+k} \tag{15.26}$$

The one-step temporal difference update in (15.20) may be replaced with one based on λ-returns,

$$J_{t+1}(i_t) = J_t(i_t) + \gamma_t\sum_{k=1}^{\infty}(\lambda\alpha)^k\delta_{t+k} \tag{15.27}$$

The full sequence of one-step temporal differences is not immediately known when state i_t is visited, but (15.27) can be used to continue adjusting the current estimate of $J(i_t)$ as additional experiences (and hence, additional terms in the sum) are gathered. The basic idea of this approach is illustrated in Fig. 15.8: at each timestep, the latest temporal difference may be used to adjust the value functions for all states visited so far. In particular, k steps after time t, $J_{t+k}(i_t)$ can be incremented by $\gamma_t(\lambda\alpha)^k\delta_{t+k}$. In a practical application of this approach, it is convenient to keep track of the temporal difference discount factor $(\lambda\alpha)^k$ for each state using a so-called *eligibility trace*, which starts out at one when a state is visited and then "decays" by a factor $\lambda\alpha$ at each subsequent timestep. These ideas lead to the formulation of a temporal difference algorithm known as TD(λ), described formally below; the timestep subscripts on e and γ are omitted for simplicity.

TD(λ) Algorithm

Suppose that a stepsize γ between 0 and 1 has been chosen. Initialize J_0 arbitrarily, and let $e(i) = 0$ for all states $i \in \boldsymbol{S}$. Simulate the MDP, selecting actions u_t as prescribed by the fixed policy μ. After each transition to a new state i_{t+1}, perform the following updates:

$$
\begin{aligned}
&\delta_t = g(i_t,u_t,i_{t+1}) + \alpha J_t(i_{t+1}) - J_t(i_t) \\
&e(i_{t+1}) = e(i_{t+1}) + 1 \\
&\qquad \text{(or "replacing traces" variant: } e(i_{t+1}) = 1\text{)} \\
&J_{t+1}(i) = J_t(i) + \gamma\, e(i)\,\delta_t \text{ for all } i \in S \\
&e(i) = \lambda\,\alpha\, e(i) \text{ for all } i \in S
\end{aligned}
\tag{15.28}
$$

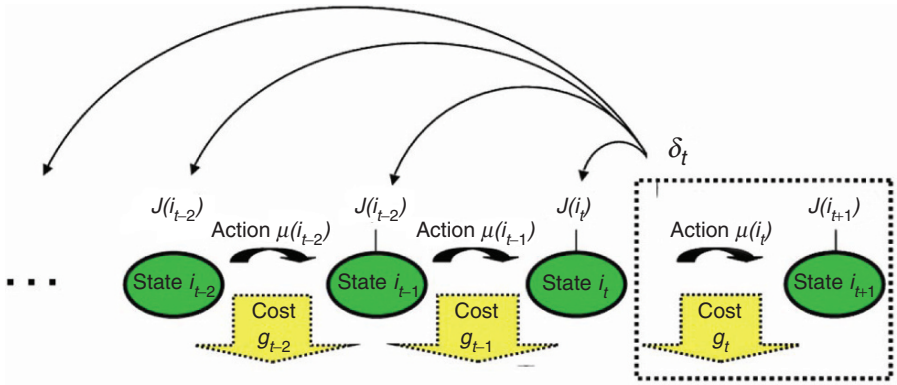


Fig. 15.8 Illustration of temporal difference learning of the value function for a policy μ. The estimate of J for every state visited so far is updated based on the temporal difference obtained from the latest timestep, $\delta_t = g_t + \alpha J(i_{t+1}) - J(i_t)$, where $g_t = g(i_t, \mu(i_t), i_{t+1})$.

The variant of TD(λ) in which the eligibility trace for a state is reset to one after each visit is called a *first-visit* or *replacing traces* method, in contrast to the *every-visit* or *accumulating traces* method in which the eligibility trace is always incremented by one. If $\lambda = 0$, this algorithm reduces to the one-step temporal difference algorithm, TD(0), described earlier. If $\lambda = 1$, it becomes an *on-line Monte-Carlo* algorithm. When the fixed stepsize γ is replaced by an appropriately decreasing sequence γ_t and every state is visited infinitely often, TD(λ) has been shown to converge to the optimal value function under fairly general conditions (Bertsekas and Tsitsiklis 1996; Dayan and Sejnowski 1994; Jaakkola et al. 1994). Several variants of TD(λ) have also been proposed; for instance, λ may be changed or "tuned" during learning to improve performance, or the eligibility traces may be updated differently.

One weakness of TD(λ) is that it learns the policy's value function, J^μ, not its Q-value. This is a significant deficiency in the model-free case, since policy improvement via (15.19) requires knowing the Q-value. In general, temporal difference methods for learning Q-values directly in the context of policy iteration can be problematic, both practically and theoretically. However, an algorithm that uses one-step temporal differences to learn the optimal Q-value directly is described in the next section.

15.9 *Q*-Learning

The temporal difference algorithms described in the previous section were stochastic approximation methods for solving the consistency equation (15.3) for a fixed policy. In contrast, Q-learning is a stochastic approximation to value iteration for solving (15.17), thus learning the optimal Q-value, Q^*, directly. This algorithm, proposed by Chris Watkins in his 1989 Ph.D. dissertation (Watkins 1989), was a major breakthrough in reinforcement learning theory. By learning Q^*, Q-learning immediately yields the optimal policy via (15.18). Another powerful feature of Q-learning is that it allows the learning agent to learn about the *optimal* policy while executing a completely *arbitrary* policy.

The basic idea of Q-learning is to update an estimate Q_t of Q^* at every timestep based on the one-step temporal difference $\delta_t = g(i_t, u_t, i_{t+1}) + \alpha \min_{v \in U(i_{t+1})} Q_t(i_{t+1}, v) - Q_t(i_t, u_t)$. The quantity $g(i_t, u_t, i_{t+1}) + \alpha \min_{v \in U(i_{t+1})} Q_t(i_{t+1}, v)$ from the latest interaction with the environment provides a new (but noisy and possibly biased) estimate for $Q^*(i_t, u_t)$, and so the stochastic approximation "nudging" approach described in the previous section is used to refine the current Q^* estimate based on this new information. This process is illustrated in Fig. 15.9 and described formally below.

Q-*Learning Algorithm*

Suppose $\gamma_t(i, u) \geq 0$ for all times t, states $i \in \boldsymbol{S}$ and actions $u \in U(i)$, and initialize Q_0 arbitrarily. Simulate the MDP under any policy, not necessarily deterministic or even stationary. After each

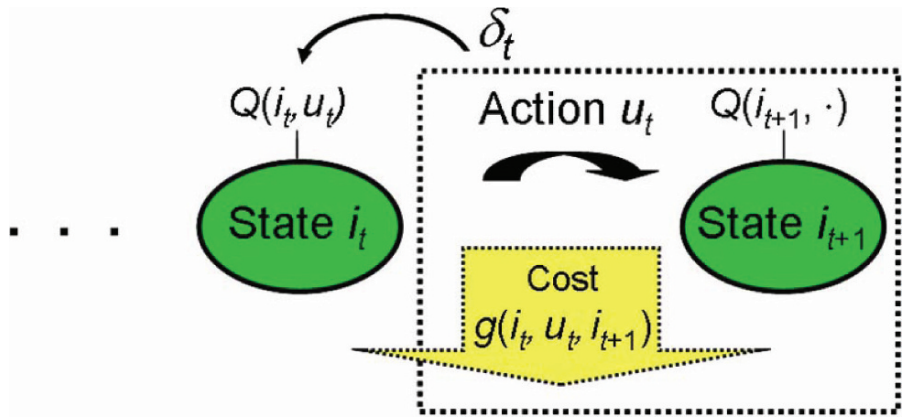


Fig. 15.9 Diagram of the Q-learning algorithm update. A one-step temporal difference based on the latest transition cost and minimum Q-value in the new state is used to update the optimal Q-value estimate for the action taken in the previous state.

transition from a state i_t to a new state i_{t+1} under action u_t, perform the following update:

$$Q_{t+1}(i_t, u_t) = Q_t(i_t, u_t) + \gamma_t(i_t, u_t)\big(g(i_t, u_t, i_{t+1}) + \alpha \min_{v \in U(i_{t+1})} Q_t(i_{t+1}, v) - Q_t(i_t, u_t)\big) \tag{15.29}$$

If the stepsizes γ_t get small in an appropriate fashion as $t \to \infty$ and each state-action pair (i,u) is visited infinitely often, then it can be shown that $\lim_{t\to\infty} Q_t = Q^*$. To guarantee that every action is explored in every state infinitely often as the learning agent interacts with the environment, Q-learning requires that a stochastic or non-stationary policy be used. On the other hand, to maintain low transition costs and perhaps speed convergence, it is common to give priority to those actions that appear best under the current Q-estimate (the so-called *greedy* actions). One strategy for achieving this balance between exploration and exploitation is to execute an *ε-greedy* policy which in state i takes an action having the smallest Q-value with probability $1 - \varepsilon$ and an action chosen randomly from $U(i)$ with probability ε, where ε is between 0 and 1. Another is to use the *Boltzmann* policy, defined for all states $i \in \boldsymbol{S}$ and all $u \in U(i)$ by

$$\pi_t(i, u) = \frac{\exp\left(-Q_t(i, u)/T\right)}{\sum_{v \in U(i)} \exp\left(-Q_t(i, v)/T\right)} \tag{15.30}$$

where $T > 0$ is called the *temperature*. An equivalent but numerically more stable alternative is to use the "advantages" $A_t(i, u) = Q_t(i, u) - \min_{v \in U(i)} Q_t(i, v)$ in place of the Q-values in (15.30). By taking $\varepsilon \to 0$ or $T \to 0$ sufficiently slowly under an ε-greedy or Boltzmann policy, one can ensure that the Q-learning algorithm converges to Q^* while also allowing the policy being executed to approach the optimal policy (see Bertsekas and Tsitsiklis 1996, p. 251). Choosing a schedule for ε or T that provides a good rate of convergence to Q^* for a particular MDP may require experimentation or tuning using a separate optimization technique. In some practical on-line applications, convergence may not be desired: choosing an appropriate fixed value of ε or T allows the learning agent to continue exploring, which may be quite useful for adapting to changes when the MDP being solved isn't truly stationary. In this case, the choice of a good fixed value for T depends on the magnitude of the values of Q and the desired amount of exploration.

15.10 Temporal Difference Control

The Q-learning algorithm described in the previous section is, in a sense, a complete solution to the classical reinforcement learning problem. It learns the optimal Q-value for an MDP, and hence the optimal policy, while offering significant flexibility in the policy actually followed. However, Q-learning uses only a one-step temporal difference return; not only does this lead to potentially high bias in an update when the next state's Q-estimate is lousy, but it also means that

only the very next experience following a visit to a state-action pair is used to improve its Q-estimate. The rate of convergence of Q-learning is therefore slower than might be achieved by using more experience (longer sequences of actions and transitions) per update. For this reason, a number of algorithms have been proposed that combine ideas from eligibility traces, policy iteration, and Q-learning with the goal of providing faster convergence; among these are Sarsa(λ) and several versions of $Q(\lambda)$. The main idea of these methods is to use eligibility traces and temporal differences to update the Q-value estimate; some also use this estimate to attempt to improve the current policy.

The two $Q(\lambda)$ algorithms described below attempt to duplicate Q-learning's ability to learn the optimal Q-value while following an arbitrary policy. The first of these is due to Watkins (1989). Motivated by an expansion of (15.17), his algorithm makes use of all N-step returns beginning from a state-action pair (i, u) in a manner similar to TD(λ). However, these returns are terminated as soon as a non-greedy action is selected.

Watkins' Q (λ)

Let the stepsizes $\gamma(i, u)$ be between 0 and 1, $e(i, u) = 0$, and initialize Q_0 arbitrarily. Simulate the MDP under any policy, not necessarily deterministic or even stationary. After each transition to a new state i_{t+1} and selection of the next action u_{t+1}, perform the following updates:

$$
\begin{aligned}
\delta_t &= g(i_t, u_t, i_{t+1}) + \alpha \min_{v \in U(i_{t+1})} Q_t(i_{t+1}, v) - Q_t(i_t, u_t) \\
e(i_t, u_t) &= e(i_t, u_t) + 1 \text{ (or "replacing traces" variant: } e(i_t, u_t) = 1) \\
Q_{t+1}(i, u) &= Q_t(i, u) + \gamma(i, u)\, e(i, u)\, \delta_t \text{ for all } i \in S \text{ and } u \in U(i) \\
e(i, u) &= \begin{cases} \lambda \alpha\, e(i, u) \text{ if } Q_t(i_{t+1}, u_{t+1}) = \min_v Q_t(i_{t+1}, v), \text{ or} \\ 0 \text{ otherwise} \end{cases} \quad \text{for all } i \in S \text{ and } u \in U(i)
\end{aligned}
\tag{15.31}
$$

For the replacing traces variant in this and other algorithms that use eligibility traces to learn Q-values, a slight enhancement is available. In addition to setting $e(i_t, u_t) = 1$, one may set $e(i_t, u) = 0$ for all $u \neq u_t$, thus terminating all N-step returns when a state is revisited. This strategy has been shown empirically to outperform both the every-visit and standard replacing traces methods (Singh and Sutton 1996).

The main weakness of Watkins' $Q(\lambda)$ algorithm is that the lengths of the backups will be very short when many exploratory actions are taken. If actions are selected based on an ε-greedy or Boltzmann policy with ε or T decreasing with time, this will often be the case early in the learning process. A second "optimistic" or "naïve" $Q(\lambda)$ algorithm, proposed by Sutton and Barto (1998) as a simpler alternative to a similar algorithm due to Peng and Williams (1996), ameliorates this problem by allowing all returns to be used – even those following an exploratory action. Although this will result in some technically incorrect updates being performed, the initial rate of learning should be higher than in Watkins' $Q(\lambda)$ algorithm, and the algorithms become nearly identical as the probability of exploratory actions is decreased.

Naïve Q(λ)

Let the stepsizes $\gamma(i,u)$ be between 0 and 1, $e(i,u) = 0$, and initialize Q_0 arbitrarily. Simulate the MDP under any policy, not necessarily deterministic or even stationary. After each transition to a new state i_{t+1}, perform the following updates:

$$\begin{aligned}
\delta_t &= g(i_t, u_t, i_{t+1}) + \alpha \min_{v \in U(i_{t+1})} Q_t(i_{t+1}, v) - Q_t(i_t, u_t) \\
e(i_t, u_t) &= e(i_t, u_t) + 1 \text{ (or ``replacing traces'' variant: } e(i_t, u_t) = 1) \\
Q_{t+1}(i, u) &= Q_t(i, u) + \gamma(i, u)\, e(i, u)\, \delta_t \text{ for all } i \in S \text{ and } u \in U(i) \\
e(i, u) &= \lambda \alpha e(i, u) \text{ for all } i \in S \text{ and } u \in U(i)
\end{aligned} \tag{15.32}$$

Note that $Q(0)$ under either Watkins' or Naïve $Q(\lambda)$ is simply ordinary Q-learning. In order for Q_t to converge to Q^*, it is generally necessary to replace the stepsizes $\gamma(i,u)$ with sequences decreasing to zero in an appropriate fashion, and to guarantee that each state-action pair is visited infinitely often; additional conditions may also be required. Under any of these methods, once Q^* is found, an optimal policy may be determined via (15.18).

A final temporal difference control algorithm, Sarsa(λ), is so-named because it uses temporal differences based on *S*tate-*a*ction-*r*eward(cost)-*s*tate-*a*ction sequences. Unlike the Q-learning or $Q(\lambda)$ algorithms, its temporal differences do not use the minimum Q-value of the next state, but rather the Q-value for the action actually selected. For this reason, Sarsa(λ) is called an *on-policy* algorithm; it learns the Q-value of the policy being executed, not the optimal Q-value. In particular, as in the Naïve Q (λ) algorithm, returns are not terminated when non-greedy actions are taken. Indeed, the N-step returns used by Sarsa(λ) can be thought of as samples of a multi-step version of the Q-value consistency equation 15.14 rather than of the Bellman Optimality equation 15.17.

Sarsa(λ)

Let the stepsizes $\gamma(i,u)$ be between 0 and 1, $e(i,u) = 0$, and initialize Q_0 and π_0 arbitrarily. Simulate the MDP, selecting actions u_t at each timestep as prescribed by π_t. After each transition to a new state i_{t+1} and selection of the next action u_{t+1}, perform the following updates:

$$\begin{aligned}
\delta_t &= g(i_t, u_t, i_{t+1}) + \alpha Q_t(i_{t+1}, u_{t+1}) - Q_t(i_t, u_t) \\
e(i_t, u_t) &= e(i_t, u_t) + 1 \text{ (or ``replacing traces'' variant: } e(i_t, u_t) = 1) \\
Q_{t+1}(i, u) &= Q_t(i, u) + \gamma(i, u)\, e(i, u)\, \delta_t \text{ for all } i \in S \text{ and } u \in U(i) \\
e(i, u) &= \lambda\, \alpha\, e(i, u) \text{ for all } i \in S \text{ and } u \in U(i)
\end{aligned} \tag{15.33}$$

and determine a new policy π_{t+1} using some function of $t+1$ and Q_{t+1}.

There are many ways that the new policy π_{t+1} may be determined; for instance, it may be an ε-greedy or a Boltzmann policy, where ε or T decreases with time. In order that $\lim_{t\to\infty} Q_t = Q^*$, the step sizes γ should be replaced with an appropriately decreasing sequence. The sequence of policies must eventually assign positive probabilities only to actions that are greedy with respect to the Q-estimate while ensuring that each state-action pair is visited infinitely often. Such a sequence of policies is called *greedy in the limit with infinite exploration* (GLIE) in Singh et al. (2000), where it is proven that one-step Sarsa, Sarsa(0), converges under GLIE policy sequences. Additional results on the convergence of optimistic policy iteration algorithms may be found in Tsitsiklis (2002).

15.11 Partially Observable MDPs

The techniques presented in this chapter are effective for solving MDPs, but in some practical applications, the exact state of the environment may not

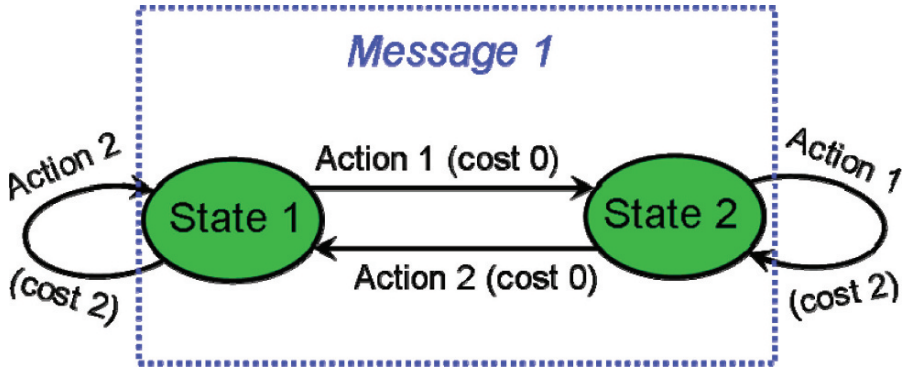


Fig. 15.10 Example POMDP. The underlying MDP has two distinct states, two available actions and deterministic state transitions represented by the arrows; costs associated with the state transitions are indicated in parentheses. However, the learning agent cannot sense the precise states but only a single message that fails to distinguish them.

easily be determined. With only an estimate of the state available, the history of past actions may influence future costs and transition probabilities, violating the Markov hypothesis. When these effects are small or the problem is otherwise well-behaved, traditional reinforcement learning methods such as Q-learning may still provide good solutions. However, in general these Partially Observable Markov Decision Processes (POMDPs) may be much more difficult than MDPs to solve (Lovejoy 1991).

To formalize the notion of a POMDP, we assume that the environment has the structure of an MDP but the learning agent perceives only an observation, or *message*, from a set $\mathbf{M} = \{m_1, m_2, \ldots m_M\}$, with the message determined according to a conditional probability distribution $P(m|i)$ for message m given the true state i. The learner will in general not have a model for the underlying MDP or even know the number of true states. Thus, while actions continue to drive state transitions in the MDP, only the costs incurred and the current message will be available to the learning agent.

One approach to solving POMDPs is to use the history of observations, actions, and costs to try to obtain an improved estimate of the true, hidden state (e.g., Chrisman 1992), but such methods can be computationally expensive and may not scale well for large problems. An alternative approach is to learn the best stochastic memoryless policy, i.e., a policy that prescribes a probability distribution over available actions based only on the immediately available message. Such policies are the natural solution to a wide class of problems including games and their economic analogs, since an intelligent opponent could adapt to exploit any fixed, deterministic policy. A simple example of a POMDP for which the optimal memoryless policy is stochastic is depicted in Fig. 15.10. Although there are two distinct states in the underlying MDP, the learning agent receives a message that fails to distinguish them. If action 1 is always taken, a cost of 2 will be incurred at every timestep following the first one. The same is true if the policy is to always take action 2. Thus, either deterministic policy will result in an average cost per timestep of 2. However, a stochastic policy that prescribes taking action 1 with probability 0.5 and action 2 with probability 0.5 will result in an average cost per timestep of 1; this is the optimal memoryless policy for the POMDP. Since stochastic policies include deterministic policies, which simply set the probability of all but one action in each state to 0, it always makes sense to look for the optimal memoryless policy for a POMDP in the class of stochastic policies.

Unlike in an MDP, where there exists an optimal policy that simultaneously produces the minimum possible value function for each state, there may not be such a policy for a POMDP. Thus, a measure of performance other than the expected sum of discounted costs (the measure used for MDPs) may be required. One such measure is given by the asymptotic average cost per timestep that a stochastic policy π achieves, as described in the example above. A method for computing estimates of Q-values related to this quantity and using them to incrementally improve a stochastic policy is described in Jaakkola et al. (1995). A modified version of this method was used in Williams and

Singh (1999) to learn stochastic policies for several problems including a matrix game, a robot navigating a maze with imperfect state sensor, and the dynamic assignment of jobs in a server queue. An alternative approach, which uses the idea of gradient descent in a parameterized stochastic policy space, may be found in Baxter and Bartlett (2000).

15.12 Learning with Function Approximation

Although the theory and algorithms described so far in this chapter have assumed that the states and actions are discrete and sufficiently small in number that their value function or Q-values could be stored and updated in "table-lookup" fashion, this may not be true for many practical control problems. For example, realistic navigation problems do not take place within a simple maze in which only four moves are possible; rather, the locations may be described via distances from certain landmarks, the terrain may be variable, and movements in any direction may be possible. In cases of very large, infinite, or continuous state and action spaces, it becomes necessary to represent value functions and Q-values via a parameterized function approximator – e.g., a linear approximation or a neural network. When the reinforcement algorithm being utilized calls for a stochastic approximation Q-value update, the function approximator's parameters are adjusted in such a fashion that the Q-value it represents for the relevant state and action is nudged moves in the direction of the new estimate.

To make this more precise, let us consider on-line learning algorithms that operate on Q-values and suppose that $Q(i,u) = f_{\mathbf{w}}(i,u)$, where $\mathbf{w}$ is a vector of parameters for the approximation function f. For instance, if f were a neural network, $\mathbf{w}$ might represent the vector of all connection weights and activation thresholds; in a polynomial approximation, it might represent the set of all coefficients. At time t, the vector $\mathbf{w}_t$ yields a Q-value $Q_t(i_t,u_t) = f_{\mathbf{w}_t}(i_t,u_t)$ and the learning algorithm produces a new estimate $\hat{Q}_{t+1}(i_t,u_t)$ for $Q_{t+1}(i_t,u_t)$. The vector $\mathbf{w}_t$ must now be incrementally changed so that $f_{\mathbf{w}_{t+1}}(i_t,u_t)$ is "nudged" towards $\hat{Q}_{t+1}(i_t,u_t)$. Here nudging is used in place of simply adjusting $\mathbf{w}_t$ to make $f_{\mathbf{w}_{t+1}}(i_t,u_t) = \hat{Q}_{t+1}(i_t,u_t)$ because the latter could unduly degrade the Q-value representations of other nearby state-action pairs. In fact, the manner in which state-action pairs (i_t,u_t) are chosen for updates can be quite important in determining whether the approximation $f_{\mathbf{w}_t}$ behaves as desired (Tsitsiklis and Van Roy 1997; Tadic 2001). Interaction with a real or simulated environment while following a GLIE policy seems to be a good approach to producing a suitable sampling of state-action pairs, but choosing an appropriate schedule for diminishing exploration may require experimentation.

Once a method for generating actions is chosen, $\mathbf{w}_t$ may be adjusted by performing a single step of gradient descent on the squared difference between the function approximator output and the new Q-value estimate:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t - \frac{1}{2}\xi_t \nabla_{\mathbf{w}_t}\left[f_{\mathbf{w}_t}(i_t,u_t) - \hat{Q}_{t+1}(i_t,u_t)\right]^2 \\ &= \mathbf{w}_t - \xi_t\left[f_{\mathbf{w}_t}(i_t,u_t) - \hat{Q}_{t+1}(i_t,u_t)\right]\nabla_{\mathbf{w}_t} f_{\mathbf{w}_t}(i_t,u_t)\end{aligned} \tag{15.34}$$

where ξ_t is a stepsize parameter between 0 and 1 and $\nabla_{\mathbf{w}} f$ is the gradient of f – the vector of partial derivatives of f with respect to the components of $\mathbf{w}$. If f is a neural network, the update of $\mathbf{w}_t$ may be performed using standard on-line backpropagation, where the association $(i_t,u_t) \mapsto \hat{Q}_{t+1}(i_t,u_t)$ is considered a training example. In a learning algorithm that produces a new Q estimate of the form $\hat{Q}_{t+1}(i_t,u_t) = Q_t(i_t,u_t) + \gamma_t\,\delta_t$, (15.34) becomes

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \xi_t\,\gamma_t\,\delta_t\,\nabla_{\mathbf{w}_t} f_{\mathbf{w}_t}(i_t,u_t) \tag{15.35}$$

Both ξ_t and γ_t must diminish towards zero in an appropriate fashion as learning continues in order for $\mathbf{w}_{t+1}$ to converge. If eligibility traces are used in the learning algorithm, as in $Q(\lambda)$ or Sarsa(λ), they must now be replaced with eligibility traces over the components of $\mathbf{w}$. A common approach is to update the vector of eligibility traces, $\mathbf{e}$, as

$$\mathbf{e}_{t+1} = \lambda\,\alpha\,\mathbf{e}_t + \nabla_{\mathbf{w}} f(i_t,u_t) \tag{15.36}$$

so that future adjustments via $\mathbf{w}_{t+1} = \mathbf{w}_t + \xi_t\,\gamma_t\,\delta_t\,\mathbf{e}_t$ will continue to update the components of $\mathbf{w}$ as new information is obtained, thus speeding learning.

It should be noted that there are very few theoretical results that guarantee the convergence of

reinforcement learning algorithms when function approximation is used, and even for the case of linear function approximation with Q-learning there are counterexamples for which the Q-values diverge to infinity (Baird 1995; Precup et al. 2001). Nevertheless, there are also many success stories demonstrating the potential usefulness of this approach, and a number of techniques and rules of thumb have been developed for handling optimal stopping and other problems that arise (Bertsekas and Tsitsiklis 1996). As is always the case in using neural networks or other function approximators, it is very important that a data representation be carefully chosen. In particular, it is desirable to derive features of the states and actions for use as inputs to the function approximator; these should be chosen so that, to the extent possible, "nearby" features lead to similar costs and state transitions. Experience and experimentation will likely be necessary to get the best results.

15.13 Applications of Reinforcement Learning

This section presents applications of reinforcement learning to three sample problems relevant to environmental science: dynamic routing of sensor data in a wireless array, control of a scanning remote sensor, and aircraft routing in an environment for which probabilistic weather hazard information is available.

15.13.1 Dynamic Routing in Wireless Sensor Arrays

Wireless arrays of small, battery or solar-powered *in situ* sensors are beginning to provide an exciting new technology for environmental science research. Unlike many traditional research deployments, such arrays can be quickly and easily deployed in an *ad hoc* fashion to measure biogeochemical and other environmental processes at a high temporal and spatial resolution. In order to be most flexible and efficient, the sensors are placed in the locations of scientific interest and then self-organize their communications to relay or "hop" measurements back to a base station that records them. This approach, known as *mesh networking*, allows the sensor network to work well even when many of the sensors are out of line-of-sight with the base station (e.g., behind a tree or over the crest of a hill); moreover, it allows the use of smaller radios and lower-power transmissions than would be necessary for each sensor node to communicate directly with the base station.

Equipping a wireless sensor array to learn a good network communications structure and adapt it as conditions change can be accomplished using reinforcement learning techniques. One way to formulate the problem would be to consider it an SSPP similar to the "robot maze", where the goal is to relay a message from the originating node to the base station in the minimum number of steps. However, the sensor array communications routing problem is not really an SSPP because the connection strengths between nodes, available battery power, and network traffic are all variable and may change over time, and occasionally a node may fail altogether. These changes will alter the cost of transmission and the probability that a message will be successfully received. For example, if the shortest path to the base station is used repeatedly, some nodes may experience much more traffic than others and hence greater battery drain and more rapid failure. In order to ensure a robust network, the routing scheme should balance the transmission loads on the different nodes to the extent possible; thus, the performance metric to be optimized should incorporate not only the number of hops a message has to take, but also the traffic on the most power-limited nodes in the network.

There are many ways to formulate this problem, and we will use a fairly simple one. We choose as "states" the nodes in the network, with actions given by the choice of which target node to attempt to transmit to. If the message is received, the receiving node will include information on its battery power and its current minimum Q-value in the acknowledgement signal; in practice, this acknowledgement of reception may be obtained by "overhearing" the message's retransmission. We define the cost of a transmission to be 1/(fraction of battery power remaining) for the *receiving* node. If no acknowledgement signal is received in a certain time period, then the transmission is deemed to have failed, presumably because the targeted receiving node is out of range, has insufficient battery power, or is busy with another transmission. In this case, the state returns to the sending node, which is then also

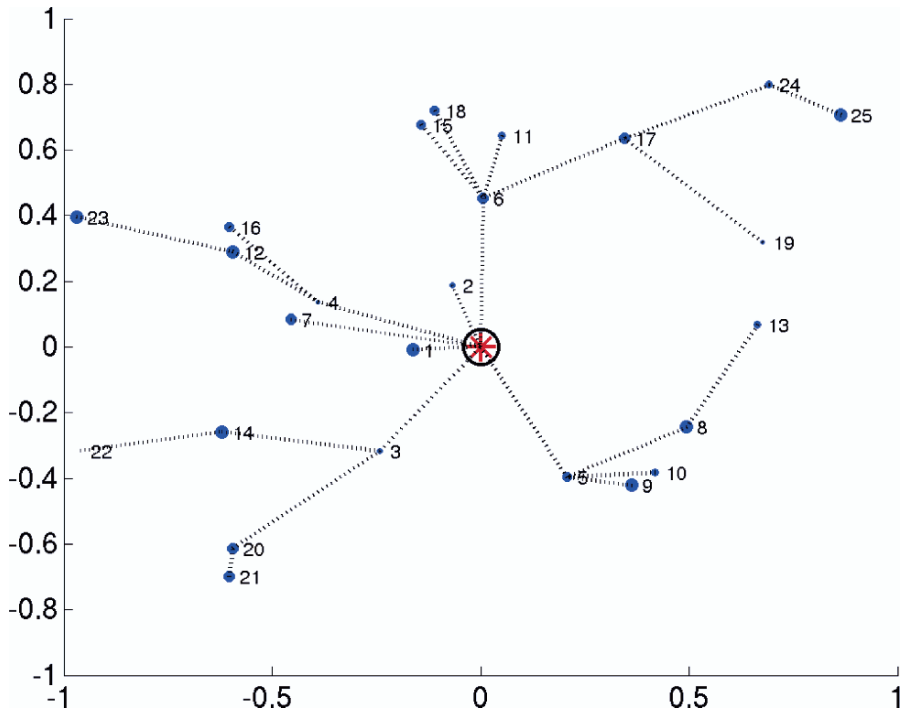


Fig. 15.11 Random deployment of a wireless sensor array consisting of 25 nodes, with the base station at the origin. The size of the blue circle representing each node is proportional to the frequency with which messages originate at that node. Nodes are numbered in order from closest to farthest from the base station, and the shortest-path routes (those with the lowest mean number of transmissions required to get a message to the base station) are indicated by dotted lines.

considered the receiving node for the purpose of determining the cost of the transmission. This formulation is not truly an MDP because the transition probabilities and costs are not stationary; rather, they change in time based on the history of past transmissions. Nevertheless, we will demonstrate that a straightforward application of Q-learning is capable of finding good policies for routing messages through the network and adapting the policies as conditions change.

We create a scenario for testing this approach via simulation by randomly placing 25 sensor nodes in a 2×2 unit domain with the base station in the center, as shown in Fig. 15.11. Each node is assigned a randomly-chosen probability for producing a message at each timestep. This probability accounts for the fact that sensor nodes may employ adaptive reporting strategies, with each node transmitting measurements more frequently when "interesting" phenomena are detected at its location. To model uniform time-based reporting, these probabilities could be set to 100%, or a regular reporting schedule could be implemented for each node. The probability that a message will be received by a target node and successfully acknowledged is determined by a function of the distance between the two nodes, as shown in Fig. 15.12. (This function was chosen for the purpose of this didactical example and does not necessarily represent the performance of any specific radio technology.) Each transmission reduces the sending node's remaining battery power by a fixed amount, and we assume that each battery begins with the capacity for sending 100,000 messages. Each node maintains a set of Q-values that are used to determine the transmission policy as described below and are updated after each transmission based on information received from the acknowledgment signal, or from the sending node if no acknowledgment is received.

The Q-learning algorithm described in Section 15.9 is a good choice for solving this on-line learning problem because information from the next state (node) is naturally available via the acknowledgment signal, whereas the multi-step backups needed

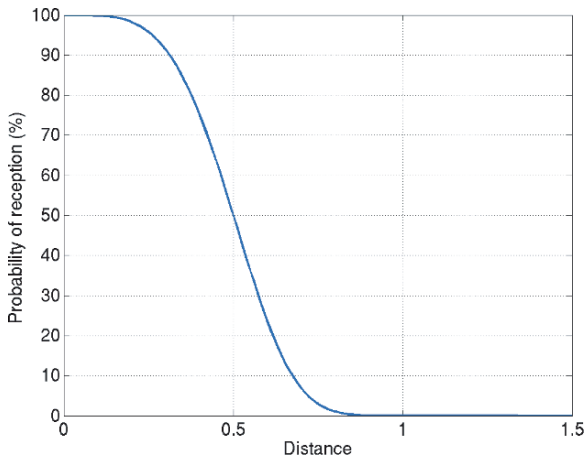


Fig. 15.12 Probability of a successful transmission from one sensor node to another, including receipt of an acknowledgement, as a function of the distance between them.

by $Q(\lambda)$ or Sarsa(λ) would require additional transmissions. We use the Boltzmann exploration policy from (15.30) with advantages $A_t(i,u) = Q_t(i,u) - \min_{v \in U(i)} Q_t(i,v)$ replacing Q-values and a choice of $T = 0.05$. Note that smaller values of T would tend to lead to fewer random "exploratory" transmissions and less ability to adapt to changing conditions, while larger values of T lead to more exploratory transmissions and lower network efficiency. The learning rate was chosen as $\gamma = 0.2$. Smaller values of γ would lead to more stable and possibly more accurate Q-values and a more consistent transmission strategy, while larger values allow quicker adaptation to changing conditions. The best choice of exploration and learning rates depends in part on the size of the network and the timescales of the changes that affect sensor reporting frequencies and transmission success; experimentation or tuning via simulation may be necessary to obtain optimal results.

Figure 15.13 shows comparative results from the fixed shortest-path routes (those with, on average, the smallest number of transmissions required to reach the base station) indicated by the dotted lines in Fig. 15.11 and the online adaptive Q-learning strategy described above. For each episode of the simulation, an originating node is chosen at random according to the origination probabilities, the message is transmitted through the network according to the current routing policy, and the episode ends when the message is received at the base station. The left panels show a running average of the total number of transmissions required for each message on a log-log plot, and the right panel shows the battery power remaining for each of the 25 sensors as a function of the cumulative number of messages received at the base station. While the number of transmissions per message remains constant over time for the static routing strategy (with some fluctuation due to the randomness in the simulation) and the battery power for each node decreases linearly with time, the results for the adaptive method shows an initial reduction in the average number of transmissions for each message as the network learns a good strategy and then an eventual increase as the network reorganizes to reduce the strain on the most-utilized nodes. The static routing strategy results in the failure of node 6 due to exhausted battery power after 213,967 messages are received at the base station, whereas the adaptive method successfully transmits 507,715 messages before failure. Thus, the adaptive approach is able to lengthen the lifetime of the entire network by more than a factor of two for this scenario. Of course, these results represent only single simulation runs using each method, and results can be expected to vary somewhat due to the random elements of the simulation.

Traditional mesh networks reorganize their routing topology periodically to adapt to changing conditions or the failure of some of the network nodes, and so avoid the stark failure illustrated here. A standard technique for reorganizing is to send a flood of transmissions, or "beacon", from the base station down to the nodes, which results in additional drain on the node batteries. The adaptive Q-learning method achieves the same objective while avoiding the extra transmission costs for periodic wholesale network reformation. A precise comparison of the two methods would require a more realistic simulation or *in situ* test for a specific application, but it is clear from this example that a technique based on reinforcement learning has the potential to produce efficient adaptive network routing strategies.

15.13.2 Adaptive Scanning Strategy for a Remote Sensor

An optimal control problem that arises frequently in environmental science research is how to target

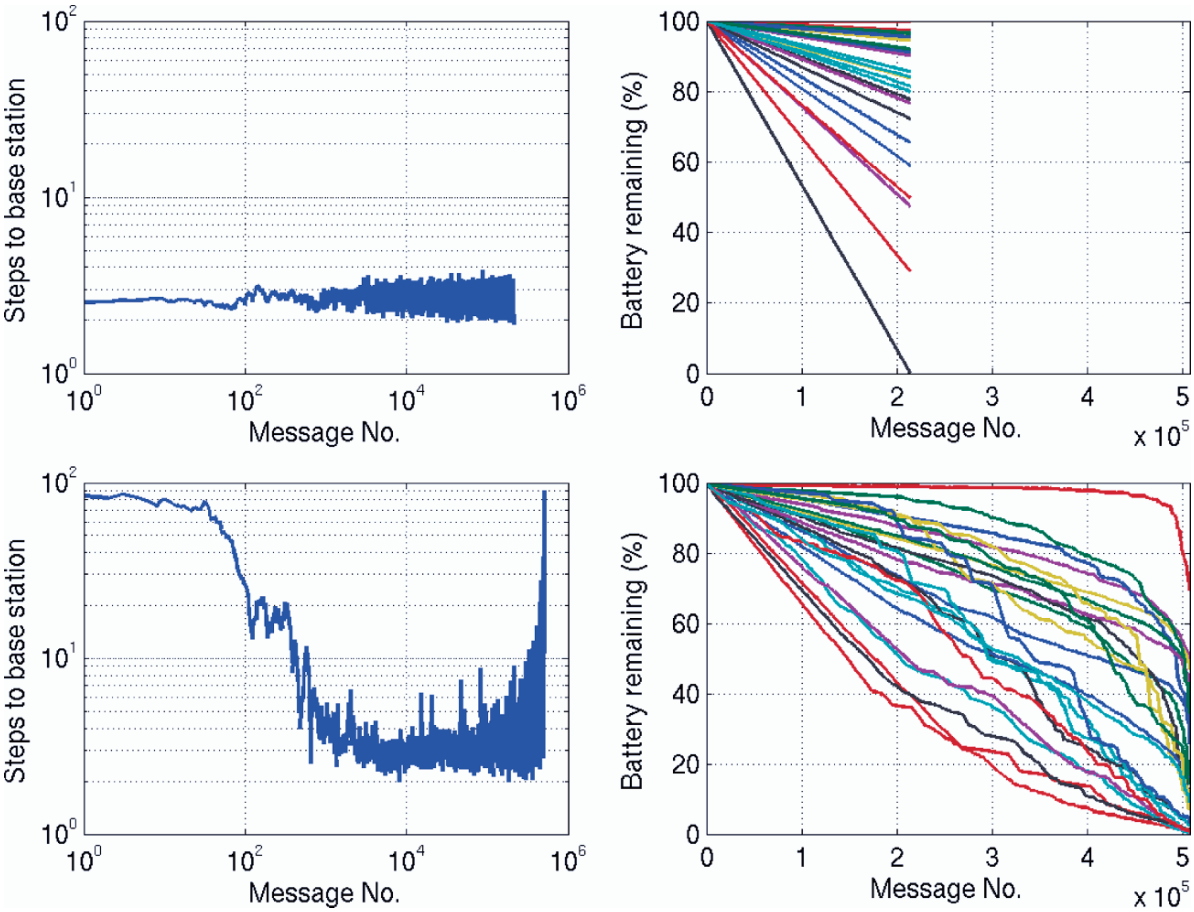


Fig. 15.13 (Top row) Results from using the shortest path routes indicated by the dotted lines in Fig. 15.11. (Bottom row) results from using dynamic routing based on Q-learning as described in the text. The plots on the left show a running average over 20 messages of the number of transmissions utilized for the message to reach the base station. The right hand side shows the remaining battery power in each node as a function of the message number. In order from the least to greatest battery power remaining after message number 2×10^5, the traces in the upper plot are for nodes 6, 5, 3, 4, 17, 20, 12, 24, 8, 14, 7, 23, 1, 9, 25, 21, 16, 18, 15, 13, 10, 11, 19, 2, and 22, respectively, and in the lower plot are for nodes 5, 6, 2, 1, 3, 17, 14, 7, 4, 8, 20, 24, 9, 16, 12, 23, 25, 21, 10, 15, 18, 13, 11, 19, and 22, respectively.

observations to produce the greatest benefit in understanding environmental processes, supplying data for modeling or forecasting, or providing timely warning of hazardous conditions. For example, adaptive scanning is employed by spaceborne remote sensors to improve their ability to capture significant meteorological phenomena using limited resources (Atlas 1982). Developers of the National Science Foundation's Collaborative Adaptive Sensing of the Atmosphere (CASA) X-band radar network plan to use coordinated adaptive sensing techniques to better capture precipitation events and detect flash floods or tornadoes (McLaughlin et al. 2005). And the United States' operational NEXRAD Doppler radars employ a number of volume coverage patterns (VCPs) appropriate to different weather scenarios, which may be selected automatically or by a human operator. Reinforcement learning can be used to develop an adaptive sampling strategy for a scanning remote sensor that balances a need for enhanced dwell-times over significant events with maintaining adequate temporal or spatial scanning coverage to quickly capture new developments.

In particular, we consider the problem of controlling a scanning remote sensor such as a radar, lidar, or scanning radiometer capable of taking measurements in one of four directions: north, east, south or west. At each timestep, the sensor may rotate clockwise,

maintain its current orientation, or rotate counterclockwise; it then detects the state of the atmosphere in the new sector, characterized as "clear" (e.g., no clouds), "developing" (e.g., significant clouds), or "hazardous" (e.g., tornadic supercell). The sensor memory stores the most recent observations made in each sector and the elapsed times since they were last scanned. This information, coupled with the current observation, comprise the system state and will be the basis for the decision of which direction the sensor should rotate in the next timestep. In order to ensure a minimal temporal coverage and also limit the number of possible states in the MDP, the sensor reverts to a standard clockwise scan strategy whenever the elapsed time for any sector reaches 10 or more. Thus, the state is determined by the last observation in each of the four directions and the elapsed time in the directions not currently being observed, for $3^4 \cdot 12^3 = 139{,}968$ possible states. Of course, not all of these states are actually "reachable" – for instance, no two sectors can have the same elapsed times in practice. Moreover, symmetry can be used to further reduce the effective number of states; in essence, we assume that the sensor only rotates clockwise but the order of the sectors can be reversed.

Our goal is to train the sensor to quickly detect significant events (e.g., hazardous weather) and dwell on them to the degree possible to improve the quality of their characterization and reduce the lead time for warnings to the public. To achieve this, we impose a cost function that at each timestep charges a penalty for each sector in which there is hazardous weather. The amount of the penalty is based on the elapsed time, Δt, since the sector was scanned: for $\Delta t = 0$, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 timesteps, we define $g(\Delta t) = 0, 1, 2, 3, 4, 5, 6, 8, 10, 12, 14, 17, 20$, respectively. Note that Δt is encoded in the MDP's state, so the cost function g is a deterministic function of the state. We choose a value of α close to 1 so that future costs are not discounted too much, say $\alpha = 0.995$. Finally, the manner in which the weather, w_t, changes with time may be specified via a conditional probability matrix such as

$$P(w_{t+1}|w_t) = \begin{bmatrix} 0.98 & 0.17 & 0.00 \\ 0.02 & 0.79 & 0.08 \\ 0.00 & 0.04 & 0.92 \end{bmatrix} \tag{15.37}$$

which may be interpreted as follows: if $w_t = 1$ ("clear"), the probability is 98% that $w_{t+1} = 1$, 2% that $w_{t+1} = 2$ ("developing"), and 0% that $w_{t+1} = 3$ ("hazardous"); if $w_t = 2$ the probabilities are 17%, 79%, and 4%; and if $w_t = 3$ the probabilities are 0%, 8%, and 92%, respectively. If the conditional probability matrix P doesn't change with time, a standard result from Markov theory tells us that at any later time $t + k$,

$$P(w_{t+k}|w_t) = P(w_{t+1}|w_t)^k \tag{15.38}$$

where the exponent by k, a positive integer, represents the matrix multiplied by itself k times. In fact, as $k \to \infty$, we can compute

$$\lim_{k\to\infty} P(w_{t+k}|w_t) = \lim_{k\to\infty} P(w_{t+1}|w_t)^k = \begin{bmatrix} 0.85 & 0.85 & 0.85 \\ 0.10 & 0.10 & 0.10 \\ 0.05 & 0.05 & 0.05 \end{bmatrix} \tag{15.39}$$

showing that for the choice of P given by (15.37) the weather probability distribution will asymptotically reach the hypothetical climatological averages: "clear" 85% of the time, "developing" 10% of the time, and "hazardous" 5% of the time, regardless of the initial weather conditions.

We have now specified a model of the MDP, which comprises all the information needed to simulate the sensor observations, actions, state transitions and costs in order to use Q-learning, $Q(\lambda)$ or Sarsa(λ) to find the optimal Q-value and hence the optimal scanning policy. However, the large number of states and the relative rareness of hazardous weather means that these methods can be quite slow and very sensitive to the choice of learning rates and exploration strategy. The computational requirements could be reduced by employing state aggregation – i.e., grouping the elapsed times into categories such as "short", "long" and "very long" – to reduce the effective number of states, or by using function approximation, but either reduces the problem to a POMDP and may result in learning a good but not optimal policy. A faster and more accurate alternative is to utilize the MDP model in value iteration, solving (15.6) directly and then computing the optimal policy using (15.12). For every initial state and action, we compute the probability distribution over weather conditions in all four directions via (15.38) using the last recorded observations and the elapsed times. These in turn determine the

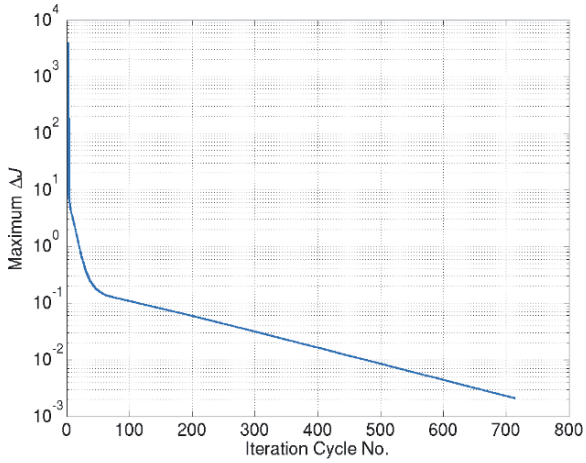


Fig. 15.14 Maximum change in the optimal value function estimate, J, after each cycle of updates during the value iteration process used to solve the remote sensor adaptive scanning problem. Because the estimate is initialized as $J = 0$, the changes are always positive.

probabilities of the three possible state transitions for each action (the new state is given by the deterministic elapsed times and random weather in the new scanning direction) and the mean cost of each transition (based on the random weather in all the other sectors). The value function J is initialized as 0 for all states, so that value iteration causes a monotonic increase in J for all states; if an elapsed time greater than 12 in any sector occurs, the value of that "state" is taken as $20(1-\alpha)^{-1} = 4{,}000$ for the purpose of computing the right hand side of (15.6). Figure 15.14 shows the rate of convergence of the value iteration, as measured by the maximum change in the value function, $\max_{i \in S} |J_{t+1}(i) - J_t(i)|$, after each pass through all states. After rapid initial progress, the rate of convergence becomes exponential.

To evaluate the policy obtained from the final J-iterate via (15.12), separate 10^7-step simulations were performed for both the learned policy and a standard scan that simply rotated clockwise at every timestep. The average costs per timestep were found to be 0.16 and 0.30, respectively, showing that the learned policy improved performance by nearly a factor of two over the standard scan. In addition, for each occurrence of "hazardous" weather in any sector and at any timestep, the total number of timesteps elapsed since the sector containing it was last scanned were tabulated. As suggested by (15.39), "hazardous" weather occurred about 5% of the time, meaning that about 2×10^6 instances occurred in the four sectors over the course of each simulation. The results in Fig. 15.15 show that an elapsed time of 0 – the ideal value – was achieved about 2.5 times more frequently by the learned policy than the standard scan, while elapsed times of 1, 2 or 3 occurred less frequently. However, the learned policy did very occasionally, about 2.5% of the time, yield elapsed times of 4 or greater. The tradeoff between increased dwell over sectors with hazardous weather and the potential for occasional instances of long elapsed times is determined by the cost function, g, which can be altered to obtain different behavior.

Of course, this formulation of the scanning problem solved here is significantly simplified relative to those often encountered in practice, where weather or another phenomenon being measured is correlated from sector to sector and the scan strategy includes selection of elevations as well as azimuths. Nevertheless, this example provides a suggestion of how reinforcement learning might be used to help develop improved observing strategies.

15.13.3 Aircraft Routing Using Probabilistic Forecasts

An important goal of meteorology and environmental science is to provide reliable forecasts to aid planning and decision making. For instance, the U.S. Federal Aviation Administration's Joint Planning and Development Office (JPDO) has developed a Next Generation Air Transportation System vision (JPDO 2006) that requires probabilistic forecast grids to guide the routing of aircraft around potentially hazardous weather and improve the safety and efficiency of the National Airspace System (NAS).

The specific problem we consider is how to use probabilistic forecast grids at various lead times to plan aircraft routes that balance a desire to conserve fuel use with the need to avoid potential accidents. We focus on the problem of routing a single aircraft to a target airport, though in the NAS the safe separation of multiple aircraft must be considered as well. For the sake of simplicity, we use only 2-D Cartesian probabilistic forecast grids of aviation hazards (e.g., thunderstorms or turbulence) at a specified flight

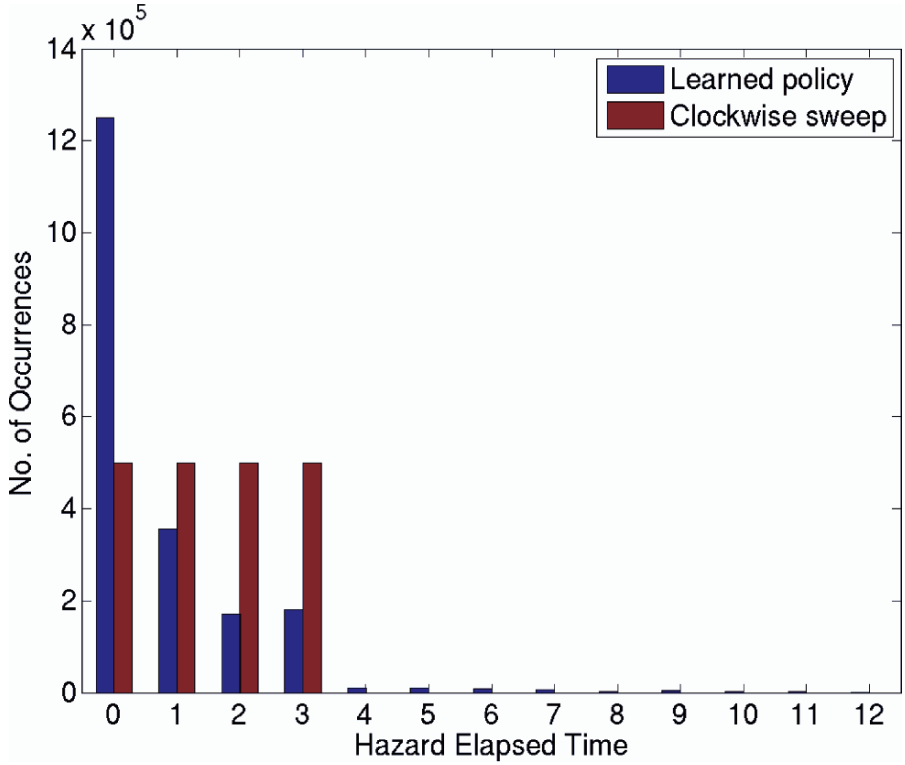


Fig. 15.15 Results from 10^7-step simulations of the scanning remote sensor using the optimal policy determined via reinforcement learning (blue) and a standard clockwise rotation at each timestep (red). The histograms show the total number of occurrences in all sectors and timesteps of the elapsed time since hazardous weather was scanned, where lower elapsed times are better; hazardous weather existed in each sector about 5% of the time, so the total number of occurrences was about 2×10^6 for each simulation.

level, but the approach illustrated here easily generalizes to 3-D grids in an appropriate map projection. The balance between fuel use and accident risk will depend on factors like the type of aircraft being flown; for instance, large aircraft are less fuel efficient but may be better equipped to mitigate hazards posed by atmospheric turbulence, icing, lightning or windshear. For a given flight, the probabilistic weather grids must be mapped to an assessment of the risk of a significant hazard to the aircraft's operation, which may be naturally quantified in terms of its potential economic impact to the airline; this allows it to be compared with other risks and impacts to form a basis for decision making. The MDP's state is given by the time and location of the aircraft. For the purpose of this example, a timestep is 5 min, the aircraft flies at a constant speed of 480 knots (40 nautical miles, nmi, per timestep), the cost of fuel is \$500 per timestep, and the weather forecast grids at each time t are scaled into the probability f_t of an accident costing \$2,500,000 for every timestep (5 min) spent in those conditions. For planning purposes, the cost of each flight segment is estimated as the sum of the fuel cost and the cost of an accident times its probability (that is, its "expected cost"). To be precise, the mean cost incurred at time t along a one-timestep flight segment from $\mathbf{x}_t$ along a vector $\mathbf{r}$ having length 40 nmi is given by

$$\bar{g}\left(\{\mathbf{x}_t, t\}, \mathbf{r}, \{\mathbf{x}_t + \mathbf{r}, t+5\}\right) = \$500 + \$2{,}500{,}000 \int_0^1 f_t\left(\mathbf{x}_t + \tau\,\mathbf{r}\right) d\tau \quad (15.40)$$

More complex versions of (15.40) could be developed to incorporate the cost of a delay after the flight exceeds a certain duration, or probabilities of multiple types of accidents or incidents and their associated costs due to, for example, taking an aircraft out of service for inspection or repair, providing worker's

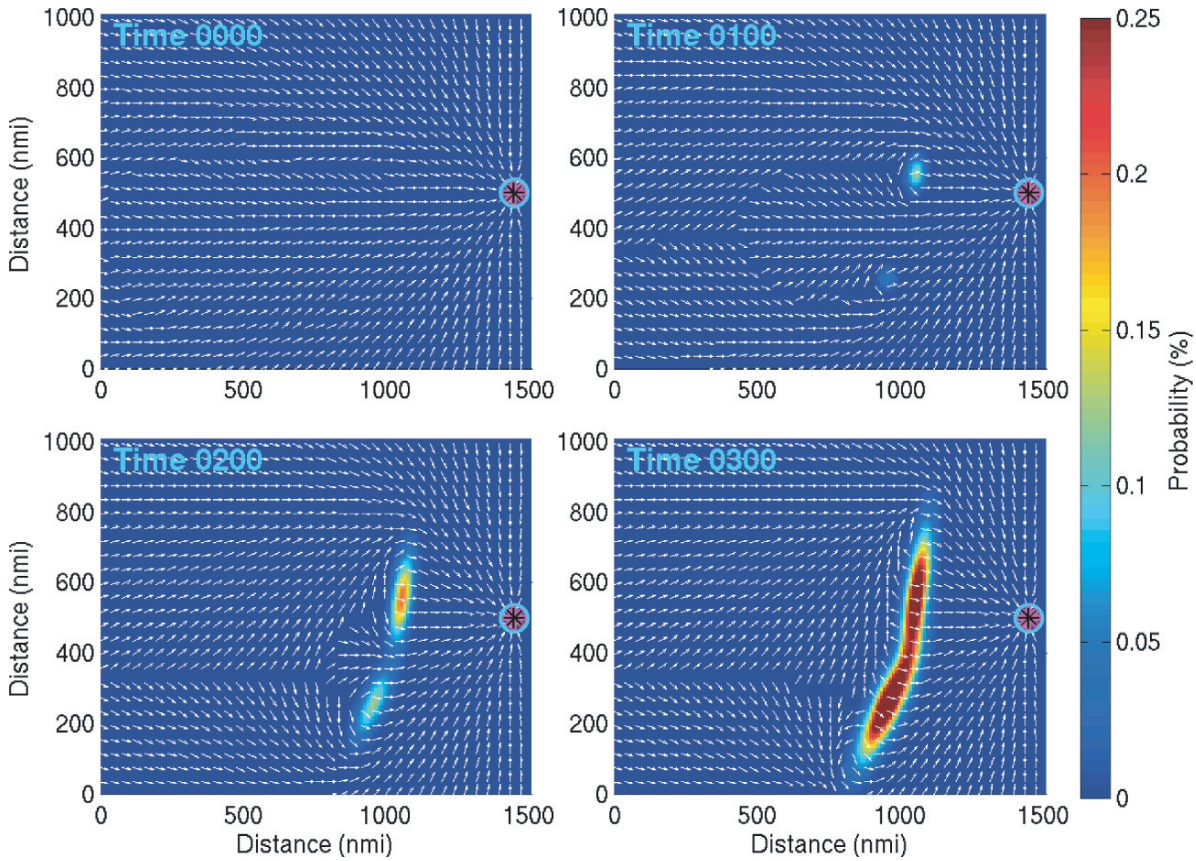


Fig. 15.16 Weather hazard probability forecasts and learned route vectors for the aircraft routing problem over a period of 3 h. The color scale gives the probability that an accident costing $2.5 million will occur during 5 min of flight through the conditions forecast at that location and time, and the destination airport is depicted on the east side of the domain.

compensation for an injured flight attendant, settling a passenger injury claim, or sustaining damage to the airline's reputation that reduces future ticket sales. An example set of forecasts f_t shown for $t = 0$, 1, 2 and 3 h (times 0000, 0100, 0200 and 0300) is given by the color-scaled grids in Fig. 15.16. This example depicts a scenario in which a line of thunderstorms is forecast to develop and block a direct route from the west to an airport in the east. Grids of f_t are available at 5 min intervals between 0 and 3 h, after which we assume the forecasts don't change; that is, we take $f_t = f_{0300}$ for $t \geq 0300$.

The aircraft routing problem as defined above is an SSPP with states given by the aircraft's position and the time, actions given by the direction of travel, and deterministic state transitions accompanied by costs prescribed by (15.40). Because the possible states and actions are not discrete but occupy a continuum, it is necessary to employ function approximation in solving this MDP. We use a lookup table representation of the value function on a grid $\{(i\,\hat{\mathbf{x}},\, j\,\hat{\mathbf{y}}) \mid i = 0,\, \ldots,\, 37;\, j = 0,\, \ldots,\, 25\}$, where $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are vectors of length 40 nmi pointing east and north, respectively; the value function at intermediate locations is estimated by linear interpolation. More precisely, the value function for the state $\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}},\, t\}$, where t is a non-negative multiple of 5 min and a and b are real numbers with $0 \leq a \leq 37$ and $0 \leq b \leq 25$, respectively, is approximated by

$$\begin{aligned}
&\tilde{J}(\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}}, t\}) \\
&\quad = (\lceil a \rceil - a)\,(\lceil b \rceil - b)\;\; J\left(\{\lfloor a \rfloor\,\hat{\mathbf{x}} + \lfloor b \rfloor\,\hat{\mathbf{y}}, t\}\right) \\
&\qquad + (\lceil a \rceil - a)\,(b - \lfloor b \rfloor)\;\; J\left(\{\lfloor a \rfloor\,\hat{\mathbf{x}} + \lceil b \rceil\,\hat{\mathbf{y}}, t\}\right) \\
&\qquad + (a - \lfloor a \rfloor)\,(\lceil b \rceil - b)\;\; J\left(\{\lceil a \rceil\,\hat{\mathbf{x}} + \lfloor b \rfloor\,\hat{\mathbf{y}}, t\}\right) \\
&\qquad + (a - \lfloor a \rfloor)\,(b - \lfloor b \rfloor)\;\; J\left(\{\lceil a \rceil\,\hat{\mathbf{x}} + \lceil b \rceil\,\hat{\mathbf{y}}, t\}\right)
\end{aligned} \tag{15.41}$$

Here J denotes the value function at a grid point, the ceiling function $\lceil a \rceil$ represents the smallest integer greater than or equal to a, and the floor function $\lfloor a \rfloor$ represents the largest integer less than or equal to a. The optimal value function, J^*, may be found using value iteration, with the right hand side of (15.6) approximated by the minimum over directions $0°, 10°, 20°, 30°, \ldots, 350°$, the transition costs determined by (15.40), and the value function of the new state approximated by (15.41). We first perform value iteration to find J^* for the static SSPP with time "frozen" at $t = 0300$. In order to speed convergence, we initialize J as the cost of flying directly to the destination airport in the absence of weather hazards (which is easily computed from the distance) and then update J at grid points chosen in order of increasing distance from the airport during each iteration. Next we reduce t by 5 min intervals and perform just one sweep through all grid points, computing J^* for that time using (15.6). The reason that only one iteration at each time is now sufficient is that the optimal value function for the next time – and, therefore, the next state – has already been determined. These calculations are continued until $t = 0$. Finally, the learned policy for each state (position and time) may be computed using (15.12), that is, we choose a policy μ such that

$$\begin{aligned} &\bar{g}\left(\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}}, t\}, \mu(\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}}, t\}), \{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}} + \mathbf{r}_\mu, t+5\}\right) \\ &\quad + \tilde{J}^*\left(\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}} + \mathbf{r}_\mu, t+5\}\right) \\ &= \min_{\{\mathbf{r}\,|\,\|\mathbf{r}\|=40\text{ nmi}\}} \Big[\bar{g}\left(\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}}, t\}, \mathbf{r}, \{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}} + \mathbf{r}, t+5\}\right) \\ &\quad + \tilde{J}^*\left(\{a\,\hat{\mathbf{x}} + b\,\hat{\mathbf{y}} + \mathbf{r}, t+5\}\right)\Big] \end{aligned} \tag{15.42}$$

where $\tilde{J}^*$ represents the linear interpolation of the learned optimal value function and $\mathbf{r}_\mu$ is shorthand for the displacement traveled in 5 min of flight in the direction specified by the policy μ. The learned policy μ may not strictly be optimal due to the use of function approximation, under which Bellman's Equality is not guaranteed to hold. Nevertheless, the policy should be near-optimal if the grid on which the value function J is represented has an appropriate scale. The vectors in Fig. 15.16 show the learned policy determined via (15.42) for each point on the 40 nmi grid at $t = 0000$, 0100, 0200 and 0300, where again for simplicity we have limited consideration to directions specified at 10° increments.

The learned policy shown in Fig. 15.16 may be interpreted as follows. At each time shown, the vectors show the direction that an aircraft starting or continuing from that location should take based on the sequence of probabilistic forecasts of future hazards. The optimal route is determined by following this "flow" as it changes at each subsequent time. Note that the domain has been chosen so that an aircraft could nearly traverse its width (1,480 nmi) in 3 h. At time 0000, the vectors in the western 2/3 of the domain are already adjusting routes to avoid the weather hazard that is forecast to develop. By time 0100, the western region shows three distinct routing strategies depending on location: go to the north of the developing line of storms, south of it or, if within about 500 nmi, fly through the gap. By 0200, the area in which aircraft are directed towards the gap has shrunk as it begins to close, and by 0300 the line of storms has fully developed and aircraft are directed around it or attempt to exit it as quickly as possible.

In addition to finding the optimal route, another important consideration may be whether to fly at all. Figure 15.17 shows the result of subtracting the learned value function from the optimal value function found in the absence of weather, a difference that represents the expected increased cost due to the weather hazard of a flight originating or continuing from the indicated location and time, assuming that it follows the prescribed optimal route from that point onwards. At time 0000, a slight increase in cost may be seen, primarily in the western part of the domain, due to the slight elongation of the optimal flight path or enhanced likelihood of an accident. By time 0100 this cost has grown substantially, particularly in the region in the west from which the aircraft is routed around the north or south of the line. The costs increase further at times 0200 and 0300, particularly for aircraft that will have to deviate substantially to avoid the storm or those that are already in it. An airline dispatcher might use this marginal "cost-to-go" information to help determine whether it would be worthwhile to delay or cancel a flight, or even divert an en route flight to another airport.

The method and results presented for this aircraft routing example can be extended in a straightforward way to more complicated scenarios. For instance, current routing in the NAS requires aircraft to fly along pre-specified "jetways" or between defined waypoints.

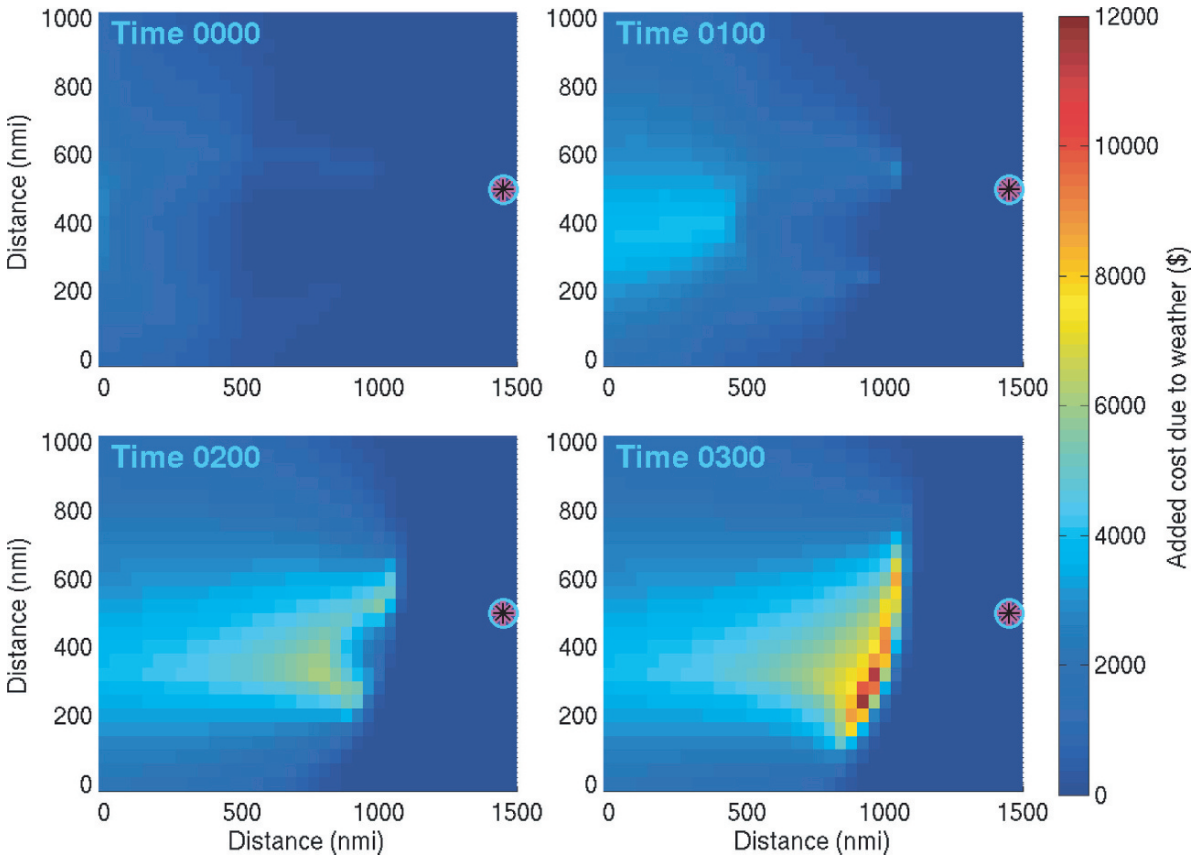


Fig. 15.17 The added (marginal) flight cost due to the weather hazard for flights originating or continuing from the indicated location and time. These values were obtained by subtracting the optimal value function, J^*, for a no-hazard situation from the optimal value function learned for the scenario depicted in Fig. 15.16 using the method described in the text.

Restricting routes to certain paths would actually substantially simplify the MDP by reducing the number of states and actions, allowing tabular representation of the value function and obviating the need for function approximation. As previously mentioned, a third dimension (altitude) can easily be added to the forecast grids and routes, and the cost function can be modified to account for the increased cost of flying at lower levels where aerodynamics are less favorable. The effect of winds, including the jet stream, can be accommodated by adding the wind vector to the aircraft's velocity at each timestep. The cost function would be specific to each aircraft type, since models have different capabilities, fuel use and ability to withstand weather and other hazards. Finally, the effects of congestion (unsafe density of aircraft) may be included as an additional hazard whose expected cost is added in (15.40). Starting with an initial forecast of congestion, the method described above may be used to find the optimal routes for all aircraft desiring to occupy the NAS, or to determine that a ground delay program or cancellation is appropriate. The optimal paths of all those flights may then be traced to revise the congestion forecast for each location and time. Then new optimal routes may be chosen based on this new cost information, and the process repeated. If this iteration is done carefully (e.g., as a relaxation method), a good overall set of routes may be obtained. Marginal costs like those shown in Fig. 15.17 may again be used to determine what flights should be delayed, cancelled or diverted to avoid undue costs and risks. Thus, reinforcement learning may have a lot to offer in designing practical solutions to this important and difficult problem. For a more detailed treatment of the topic of routing aircraft given weather hazard information, the reader is invited to consult Bertsimas and Patterson (1998), Evans et al. (2006), and Krozel et al. (2006).

15.14 Conclusion

Reinforcment learning builds on ideas from the fields of mathematics, engineering and psychology to develop algorithms that identify optimal or near-optimal control policies based on simulated or real interaction with a stochastic environment. Unlike traditional methods that require the underlying dynamical system to be modeled by a set of probability transition matrices or differential equations, reinforcement learning techniques can be applied to complex problems for which no model exists, This chapter has presented an introduction to the theory underlying reinforcement learning and the Markov Decision Processes (MDPs) to which they apply, described several practical reinforcement learning algorithms, and presented three sample applications that demonstrated the powerful potential of reinforcement learning to solve problems that arise in the environmental sciences.

Acknowledgements Satinder Singh first introduced me to reinforcement learning, and my Ph.D. advisor, Kent Goodrich, generously allowed me to explore some of its mathematical aspects for my doctoral dissertation despite its unfamiliarity to him. The idea for the aircraft routing example came from discussions with Bill Myers, whose dissertation (Myers 2000) examines how the depiction of forecast hazard data can affect humans' routing decisions. The wireless sensor array application was inspired by conversations with Lynette Laffea. Several friends and colleagues provided valuable feedback on a draft of this chapter, including Kent Goodrich, Lynette Laffea, Emily Mankin, and Matthias Steiner. I very much appreciate their assistance.

References

Atlas, D. (1982). Adaptively pointing spacebome radar for precipitation measurements. *Journal of Applied Meteorology*, *21*, 429–443.

Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In A. Prieditis & S. J. Russell (Eds.), *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 30–37). 9–12 July 1995. Tahoe City, CA/San Francisco: Morgan Kaufmann.

Baxter, J., & Bartlett, P. L. (2000). Reinforcement learning in POMDP via direct gradient ascent. *Proceedings of the 17th International Conference on Machine Learning* (pp. 41–48). 29 June–2 July 2000. Stanford, CA/San Francisco: Morgan Kaufmann.

Bellman, R. E. (1957). *Dynamic programming* (342 pp.). Princeton, NJ: Princeton University Press.

Bertsekas, D. P. (1995). *Dynamic programming and optimal control* (Vol. 1, Vol. 2, 387 pp., 292 pp.). Belmont, MA: Athena Scientific.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming* (491 pp.). Belmont, MA: Athena Scientific.

Bertsimas, D., & Patterson, S. S. (1998). The air traffic flow management problem with enroute capacities. *Operations Research*, *46*, 406–422.

Chrisman, L. (1992). Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. *Proceedings of the Tenth National Conference on Artificial Intelligence* (pp. 183–188). 12–16 July 1992. San Jose/Menlo Park, CA: AAAI Press.

Dayan, P., & Sejnowski, T. (1994). TD(0) converges with probability 1. *Machine Learning*, *14*, 295–301.

Evans, J. E., Weber, M. E., & Moser, W. R. (2006). Integrating advanced weather forecast technologies into air traffic management decision support. *Lincoln Laboratory Journal*, *16*, 81–96.

Hamilton, W. R. (1835). Second essay on a general method in dynamics. *Philosophical Transactions of the Royal Society*, Part I for 1835, 95–144.

Jaakkola, T., Jordan, M., & Singh, S. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, *6*, 1185–1201.

Jaakkola, T., Singh, S., & Jordan, M. (1995). Reinforcement learning algorithm for partially observable Markov decision problems. In G. Tesauro, D. S. Touretzky, & T. Leen (Eds.), *Advances in neural information processing systems: Proceedings of the 1994 Conference* (pp. 345–352). Cambridge, MA: MIT Press.

Joint Planning and Development Office (JPDO). (2006). *Next generation air transportation system (NGATS)—weather concept of operations* (30 pp.). Washington, DC: Weather Integration Product Team.

Krozel, J., Andre, A. D. & Smith, P. (2006). Future air traffic management requirements for dynamic weather avoidance routing. *Preprints, 25th Digital Avionics Systems Conference* (pp. 1–9). October 2006. Portland, OR: IEEE/AIAA.

Kushner, H. J., & Yin, G. G. (1997). *Stochastic approximation algorithms and applications* (417 pp.). New York: Springer.

Lovejoy, W. S. (1991). A survey of algorithmic methods for partially observable Markov decision processes. *Annals of Operations Research*, *28*, 47–66.

McLaughlin, D. J., Chandrasekar, V., Droegemeier, K., Frasier, S., Kurose, J., Junyent, F., et al. (2005). Distributed Collaborative Adaptive Sensing (DCAS) for improved detection, understanding, and prediction of atmospheric hazards. *Preprints-CD, AMS Ninth Symposium on Integrated Observing and Assimilation Systems for the Atmosphere, Oceans, and Land Surface*. 10–13 January 2005. Paper 11.3. San Diego, CA.

Myers, W. L. (2000). *Effects of visual representations of dynamic hazard worlds on human navigational performance*. Ph.D. thesis, Department of Computer Science, University of Colorado, 64 pp.

Peng, J., & Williams, R. J. (1996). Incremental multi-step *Q*-learning. *Machine Learning*, *22*, 283–290.

Precup, D., Sutton, R. S., & Dasgupta, S. (2001). Off-policy temporal-difference learning with function approximation. In C. E. Brodley and A. P. Danylok (Eds.), *Proceedings of the 18th International Conference on Machine Learning* (pp. 417–424). 28 June–1 July 2001. Williamstown, MA/San Francisco, CA: Morgan Kaufmann.

Puterman, M. L. (2005). *Markov decision processes: Discrete stochastic dynamic programming* (649 pp.). Hoboken, NJ: Wiley Interscience.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics*, *22*, 400–407.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, *3*, 211–229.

Si, J., Barto, A. G., Powell, W. B., & Wunsch, D. (Eds.). (2004). *Handbook of learning and approximate dynamic programming* (644 pp.). Piscataway, NJ: Wiley-Interscience.

Singh, S. P., & Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning*, *22*, 123–158.

Singh, S. P., Jaakkola, T., Littman, M. L., & Szepasvari, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, *38*, 287–308.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (322 pp.). Cambridge, MA: MIT Press.

Tadic, V. (2001). On the convergence of temporal-difference learning with linear function approximation. *Machine Learning*, *42*, 241–267.

Tsitsiklis, J. N. (2002). On the convergence of optimistic policy iteration. *Journal of Machine Learning Research*, *3*, 59–72.

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*, 674–690.

Turing, A. M. (1948). Intelligent machinery, National Physical Laboratory report. In D. C. Ince (Ed.). 1992, *Collected works of A. M. Turing: Mechanical intelligence* (227 pp.). New York: Elsevier Science.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*, 433–460.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Ph.D. thesis, King's College, Cambridge University, Cambridge, 234 pp.

Watkins, C. J. C. H., & Dayan, P. (1992). *Q*-learning. *Machine Learning*, *8*, 279–292.

Williams, J. K. (2000). *On the convergence of model-free policy iteration algorithms for reinforcement learning: Stochastic approximation under discontinuous mean dynamics*. Ph.D. thesis, Department of Mathematics, University of Colorado, Colorado, 173 pp.

Williams, J. K., & Singh, S. (1999). Experimental results on learning stochastic memoryless policies for partially observable Markov decision processes. In M. S. Kearns, S. A. Solla, and D. A. Cohn (Eds.), *Advances in neural information processing systems 11. Proceedings of the 1998 Conference* (pp. 1073–1079). Cambridge, MA: MIT Press.

Automated Analysis of Spatial Grids

16

Valliappa Lakshmanan

16.1 Introduction

Environmental data are often spatial in nature. In this chapter, we will examine image processing techniques which play a key role in artificial applications operating on spatial data. These AI applications often seek to extract information from the spatial data and use that information to aid decision makers.

Consider for example, land cover data. Since different locations have different types and amounts of forestry, land cover information has to be explicitly tied to geographic location. Such spatial data may be collected either through in-situ (in place) measurements or by remote sensing over large areas. An in-situ measurement of land cover, for example, would involve visiting, observing and cataloging the type of land cover at a particular location. A remotely sensed measurement of land cover might be carried out from a satellite. The remotely-sensed measurement would cover a much larger area, but would be indirect (i.e., the land coverage would have to be inferred from the satellite channels) and would be gridded (i.e., one would get only one land cover value for one *pixel* of the satellite image). Users of land-cover data often wish to use the data to recover higher-level information such as determining what fraction of a particular country is wooded – AI applications help provide such an answer, building on well understood image processing methods.

In this chapter, we will consider spatial data that are on grids, or that can be placed in grids. Spatial grids are digital in nature and arranged in rows and columns of approximately equal resolution in space. Depending on the application, there may be a time sequence of gridded data (as with weather imagery) or the temporal nature may be irrelevant (as is often the case with hazard maps).

In this chapter, we will use the standard matrix notation because it is the one most commonly used in image processing. The first dimension is the row number and the second number is the column number. Thus, (0,0) is the top-left corner and (0,1) is the first row, second column. One potentially confusing effect of the standard matrix notation is that the first dimension increases southwards. If the images are in a cylindrical equidistant projection, then latitude decreases in the first dimension and longitude increases in the second dimension.

16.2 Gridding of Point Observations

In-situ measurements may be placed on spatial grids to enable easier interpretation and analysis by automated applications. Spatial interpolation (see Fig. 16.1) is used to place point observations onto a spatial grid. If the point observations are very close together, so that the average distance between the observations is smaller than, or similar in magnitude to, the resolution of the grid resolution, a technique known as kriging may be used. If, as is more common, the observations are far apart, spatial interpolation relies on balancing the competing concerns of a smooth field and a field that matches the point observations exactly at the locations that the in-situ measurements were carried out. A Cressman analysis favors the creation of smooth fields;

Valliappa Lakshmanan (✉)
University of Oklahoma & National Severe Storms Laboratory,
120 David L. Boren Blvd., Norman, OK 73072-7327, USA
Email: Valliappa.Lakshmanan@noaa.gov

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

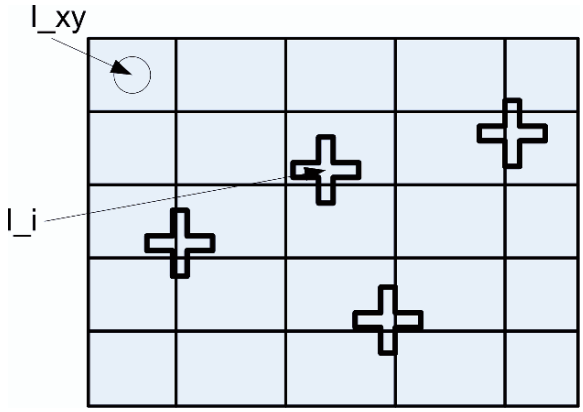


Fig. 16.1 Spatial interpolation is required to take *in situ* observations (shown by the pluses) and place them on to grids

a Barnes analysis is less smooth but attempts to match the point observations better.

Regardless of the method chosen, care should be taken that the pixel resolution is reasonable. If the chosen pixel resolution is too fine, sharp gradients in the underlying data will be smoothed away by the interpolation. If the chosen pixel resolution is too coarse, multiple observations will end up being averaged to obtain a single pixel value, resulting in degraded data quality. As a rule of thumb, it is wise to choose as the pixel resolution a large fraction (typically half) of the mean distance between the original observations.

Cressman (1959) introduced a technique of objective analysis, of interpolating observation data onto spatial grids. Consider Fig. 16.1. In Cressman analysis, the value at a pixel (x,y) is given by:

$$I_{xy} = \frac{\sum_i I_i \frac{R^2-(x-x_i)^2-(y-y_i)^2}{R^2+(x-x_i)^2+(y-y_i)^2}}{\sum_i \frac{R^2-(x-x_i)^2-(y-y_i)^2}{R^2+(x-x_i)^2+(y-y_i)^2}}$$

The impact of an observation at a pixel falls with its distance away from the pixel, so that closer observations have a much impact than observations far away. The parameter, R, determines the scaling or "the radius of influence". The larger the value of R, the more the effect of far-away points is.

Barnes analysis (Koch et al. 1983) improves on the Cressman analysis in two ways. Rather than using a polynomial weighting function, Barnes analysis uses exponential weighting functions. As our discussion on convolution filters later in this chapter will show, Gaussian functions have the nice property of providing the best possible trade-off between noise-reduction and spatial fidelity. In a Barnes scheme, the weights for interpolation are given by: e^{-r^2/σ^2} where r is the distance between the grid point and the observation.

One problem with interpolation techniques is that after gridding, even grid points at the same location as the observations have different values from the observations. This is because of the impact of farther-away points, and is often desirable in case the observation in question is faulty. If a better match to the observation point is desired, Barnes analysis allows for successive corrections. The difference between the observed values and the interpolated values at each of the observation points is interpolated onto the spatial grid and subtracted (with a fractional weight) from the result of the previous iteration. This is carried out until either a maximum number of iterations is reached or until the magnitude of the difference field is small enough. It should be kept in mind that a n-pass Barnes analysis is very sensitive to incorrect data at any of the observation points and can lead to non-smooth grids. However, it can also capture sharp boundaries much better than a 1-pass filter.

Kriging is an interpolation technique that assumes that the co-occurrence of data values can be used to gain a better interpolation. The co-occurrence is estimated by constructing a variogram – a function of correlation between pixel values against the distance between the pairs of pixels. The variogram is then used to make predictions at unobserved locations. See Oliver and Webster 1990 for more details.

16.3 Extraction of Information from Spatial Grids

It is often desired to extract information from spatial grids. For example, it might be desirable to extract from an image of land cover data, all tree-covered areas where drought conditions prevail. Or it might be desired to identify, from a weather satellite's visible channel, where a storm front is. This involves processing the image using automated applications looking for features that make it likely that an area is tree covered or undergoing drought or that the pixels of the image correspond to a storm front.

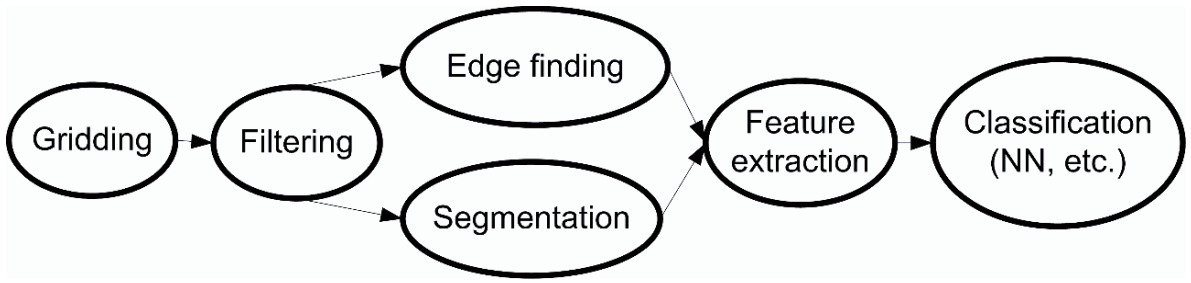


Fig. 16.2 The workflow of a typical AI application operating on spatial grids

There is a considerable body of literature and techniques for such automated analysis of images, finding predetermined objects and patterns and acting on the analysis. Image processing applies the mathematics of signal processing to two dimensions. Thus, the concept of filtering an image, reducing noise, accentuating features to make them easier to find, etc. have been the subject of much research in the electrical engineering and computer science communities. Pattern recognition follows the same approach to images but rather than simply filter images, pattern recognition approaches yield objects as results. Thus, pattern recognition adds to the image processing arsenal tools for segmentation (finding distinct areas in an image) and morphological operations (processing data on shape). Data mining is a larger field of study, of extracting information from all types of data, including data in a relational database. Knowledge discovery is a relatively new sub-discipline that in the context of spatial grids often refers to the extraction of relationships between objects that have been identified in the grids.

Image processing and pattern recognition are specific forms of data mining, concerned with processing and identifying objects in images. Although image processing and pattern recognition have been the subject of decades of research and development (and movies and TV crime shows!), these techniques work only in highly controlled environments. The presence of noise, entities that have a variety of shapes and sizes and incomplete or faulty data, pose significant problems for AI techniques. The most successful implementations of AI techniques are in such environments as factory floors, where unexpected objects are unlikely, all parts are within carefully selected parameters and incomplete data can be engineered away. AI applications in environmental science are some of the most challenging, because in many cases, it the expected size of objects is unknown, and significant artifacts pollute to the imagery presented to these techniques. The rest of this chapter presents the most mature sub-disciplines of these fields and highlights a few applications of these techniques in the automated analysis of real-time weather images.

16.3.1 Work Flow of a Typical AI Application

A typical AI application that operates on spatial imagery to find the presence or absence of some entity within the image is organized as shown in Fig. 16.2.

Preprocessing typically involves taking the spatial data and remapping it so that the grid resolution is locally uniform. If the input data are not in gridded form preprocessing may even involve placing the data in a locally uniform grid.

Uniform spatial grids are filtered to remove artifacts, noise and features that the system designer deems unnecessary or potentially disrupting to the rest of the AI process. Depending on the type of information being mined for, the gridded and filtered data are subject to either edge finding or segmentation. Edge finding finds strong gradients in the spatial data, connects them up and creates cartoons which are then subject to feature extraction and/or pattern recognition. Segmentation finds contiguous data and combines them into objects that are then subject to feature extraction and/or pattern recognition.

Features are properties computed from either the edges or the objects. For example, the eccentricity of the shape and the size and texture of the object are commonly used features. These features are then presented to a classifier such as a neural network, support vector machine or a genetic algorithm. The

output of the classifier typically indicates the presence or absence of some feature that is the goal of this AI application.

16.3.2 Markov Random Process

A basic assumption behind most image processing operations is that the image pixels can have any value, but that the value of a pixel is interwoven with the value of its immediate neighbors. This intuitive idea is formalized by assuming that the pixels of an image are generated through first-order Markov random processes. Two pixels are correlated if, and only if, they are adjacent to each other. For example, consider three adjacent pixels:

A	B	C

The pixels A and B are correlated as are the pixels B and C. The first-order Markov assumption means that the correlation between A and C is captured solely by the correlation between A–B and B–C.

This neighborhood assumption will be seen clearly in the filtering and segmentation operations. These image processing operations do not work well on fields where the pixel values of neighboring pixels are uncorrelated. Visually, such images are highly speckled, like that of a television set with no signal. Most environmental data, however, are relatively smooth spatially. Such data can be processed with image processing operations.

Gridding using spatial interpolation methods such as the Cressman and Barnes operations leads to extremely smooth fields because pixel values are obtained through interpolation of the point observations. Ideally, the grid resolution is chosen to be no more than half the maximum of the distances between every point observation and its nearest neighbor. At higher resolutions, interpolation artifacts can become evident and the neighborhood size for pattern recognition has to be made larger – a first order Markov process won't fit the data anymore. The artifacts and large neighborhood sizes can make pattern recognition harder to perform. It is recommended, therefore, that pattern recognition be performed on data gridded at a reasonable resolution.

When changing the geographical projection of a spatial grid, it is possible that, in the new projection, a couple of pixels may derive their value from the same pixel in the original image. Such repetition of pixels can also lead to spatial artifacts. It is recommended, therefore, that image processing and pattern recognition be performed on data as close as possible to the native format of the data.

Amongst image processing operations, certain operations operate only on the neighborhood of a pixel. Other operations act on the image globally or are greedy – they process as many pixels as do satisfy some predetermined constraints. Global or greedy operations are unsafe on grids where the size of a pixel varies dramatically over the image. Consider, for example, a spatial grid covering the Northern Hemisphere projected in a Mercator projection. The area covered by a pixel at the northern extremity of the image is much smaller than that of a pixel near the equator. This affects the validity of global or greedy operations since the pixels at different areas of the image are quite different. The same problem affects the processing of radar data in their original polar (actually a flattened cone, in 3D) format. Pixels closer to the radar are smaller than pixels farther away. In such situations, it is preferable to process data in a format or projection where the area of a pixel is constant. Examples of such area-preserving projections include the Albers conic and Lambert azimuthal equal-area projections. Naturally, this recommendation is at odds with the previous recommendation of performing image processing in a native format. It might be necessary to perform the operations in both a native format and in an area-preserving projection to discover what works best for a given application. In the case of radar data, the equal-area projection would be to map the radar data onto a Cartesian grid, probably one tangent to the earth's surface at the location of the radar. Repeated pixels then become problematic at long ranges. It may be necessary to try the operation both in the native polar format and in the Cartesian grid projection to discover what works best.

16.4 Convolution Filters

Intuitively, one way to reduce the noise in an image is to replace the value of a pixel with some sort of neighborhood average. Surprisingly, this very simple concept leads to very powerful local operations on images. Instead of simply using neighborhood average, more complex mathematical and statistical operations may

be performed on the set of values in the neighborhood of a pixel. For example, the median of the neighborhood values is one of the best speckle filters available. Sorting the neighborhood values and computing a weighted average of the sorted set is an excellent way to identify edges in an image. Changing the shape of the neighborhood and the weight assigned to neighborhood pixels provides the ability to identify different types of objects in images. This operation, often replacing a pixel with a weighted sum of its neighbors, is termed convolution.

Mathematically, replacing a pixel by a local average can be written as:

$$I_{xy} = \frac{1}{(2k+1)\cdot(2k+1)} \sum_{i=x-k}^{i=x+k} \sum_{j=y-k}^{j=y+k} I_{ij}$$

where I represents the image, x,y represents the pixel at row number x and column number y and k is the half-size of the neighborhood. The above equation means that for every pixel xy, we need to look in a two-dimensional neighborhood, up all the values and divide by the number of pixels. For example, if k were to be 2, we would be computing a 5×5 average around the pixel and dividing by 25. Figure 16.3 shows the effect of such local averaging operation on an infrared satellite image. Note that the jaggedness of the edges has been considerably reduced. Note also that the image is considerably smoother. Such an operation, which reduces high spatial variations in the image, is termed a smoothing operation.

16.4.1 Gaussian Filters

One need not provide the same weight to all the neighbors of a pixel. Often, a higher weight is assigned to pixels that are closer in to the center value. The above equation may be generalized as follows:

$$I_{xy} = \sum_{i=x-k}^{i=x+k} \sum_{j=y-k}^{j=y+k} W_{ij} I_{ij}$$

where the weight W essentially determines the type of operation that is performed. Technically, this operation is cross correlation. However, if one is using symmetric matrices (as we almost inevitably will be), cross correlation and convolution are the same thing. In the case of a 3×3 local average, the weight matrix is given by:

$$W = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

The local average weight matrix, often called the boxcar kernel, is a poor choice for smoothing because it can result in large spatial relocations of maxima. A better convolution kernel for smoothing is one where the weight matrix has large values in the center and smaller values around the edges. That way, values far away from the center pixel have a lesser effect on the final value than do values closer in. A Gaussian convolution kernel offers the best trade-off between spatial smoothness and the moderate variability within the image that are characteristic of local maxima. Because the features do not smudge as much, one can smooth a lot more effectively with a Gaussian kernel. In the Gaussian kernel, the values of the weight matrix are computed using this equation:

$$W_{xy} = \frac{1}{\sigma^2} e^{\frac{(x)^2+(y)^2}{-2\sigma x^2}}$$

where x and y range from −k to k. Since the Gaussian has infinite range, one needs to be careful to choose an appropriate value of k – approximately thrice the value of sigma typically works well. Note that the above matrix equation does not add up to one within the neighborhood. Therefore make sure to divide by the total weight. The higher the value of sigma, the more smoothing happens. A comparison of smoothing with the Gaussian kernel and a nearly equivalent boxcar kernel is shown in Fig. 16.4. Note that the Gaussian provides a smoother image with less smudging of maxima (the smudging of maxima is evident in the figure (top) in the greenness of the thin vertical line at the bottom left of image).

16.4.2 Matched Filters

By changing the weights in the convolution kernel, it is possible to extract a wide variety of features. For example, thin vertical lines may be extracted using the weight matrix shown below:

$$W = \frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

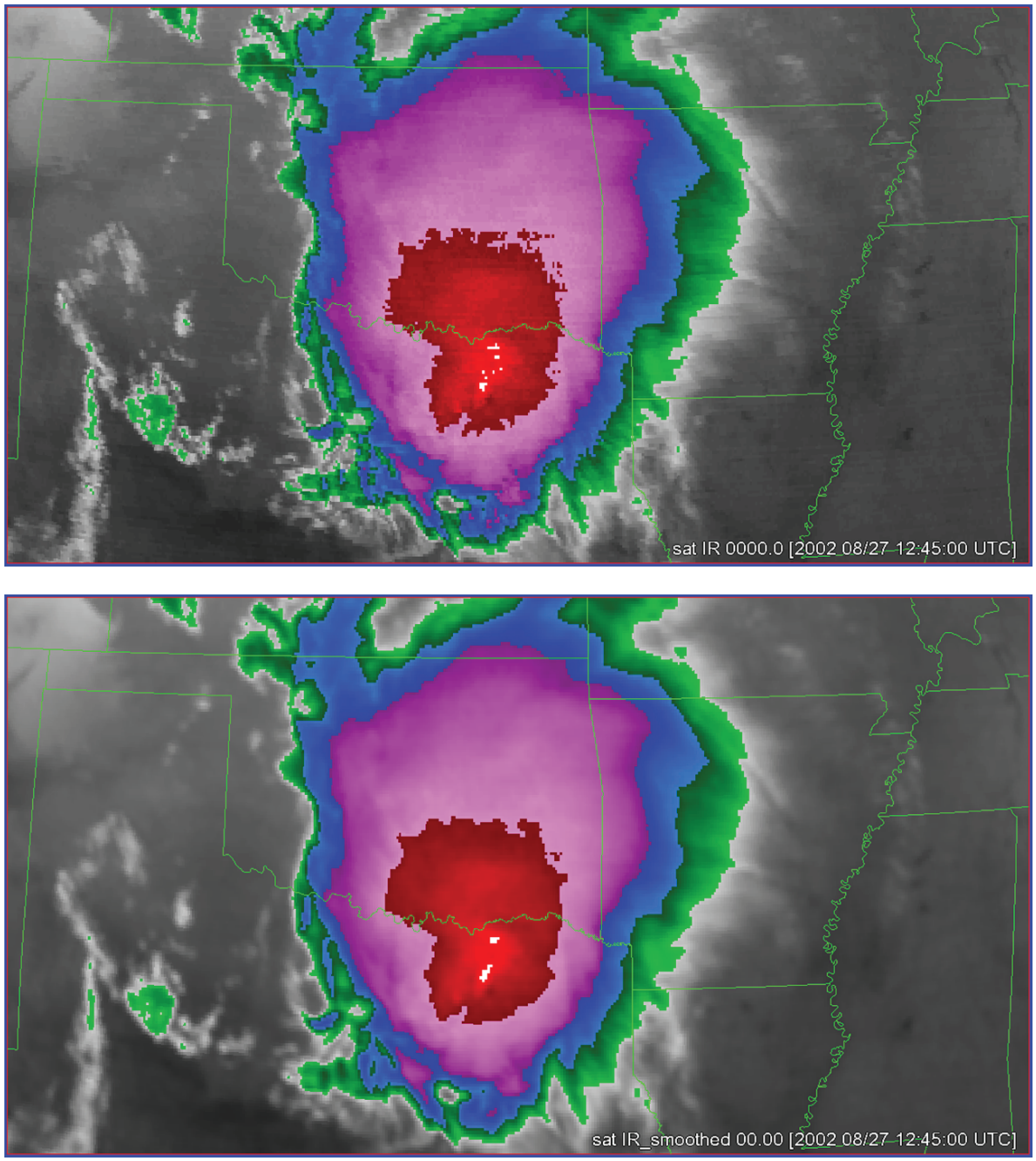


Fig. 16.3 Top: an infrared satellite image. Bottom: the same image with a 5×5 local average applied to it

Note that the above weight matrix when applied to an image results in high values, wherever there are high values to the left of low values. In areas where there is very little change, the positives and negatives canceled each other out, resulting in very small and pixel values in the in the result. The result of this operation when applied to an infrared image is shown in Fig. 16.5.

As a general principle, a convolution kernel may be used to extract features that look like it. In other words, convolution may be thought of as a matched filter operation. To avoid simply getting high values in

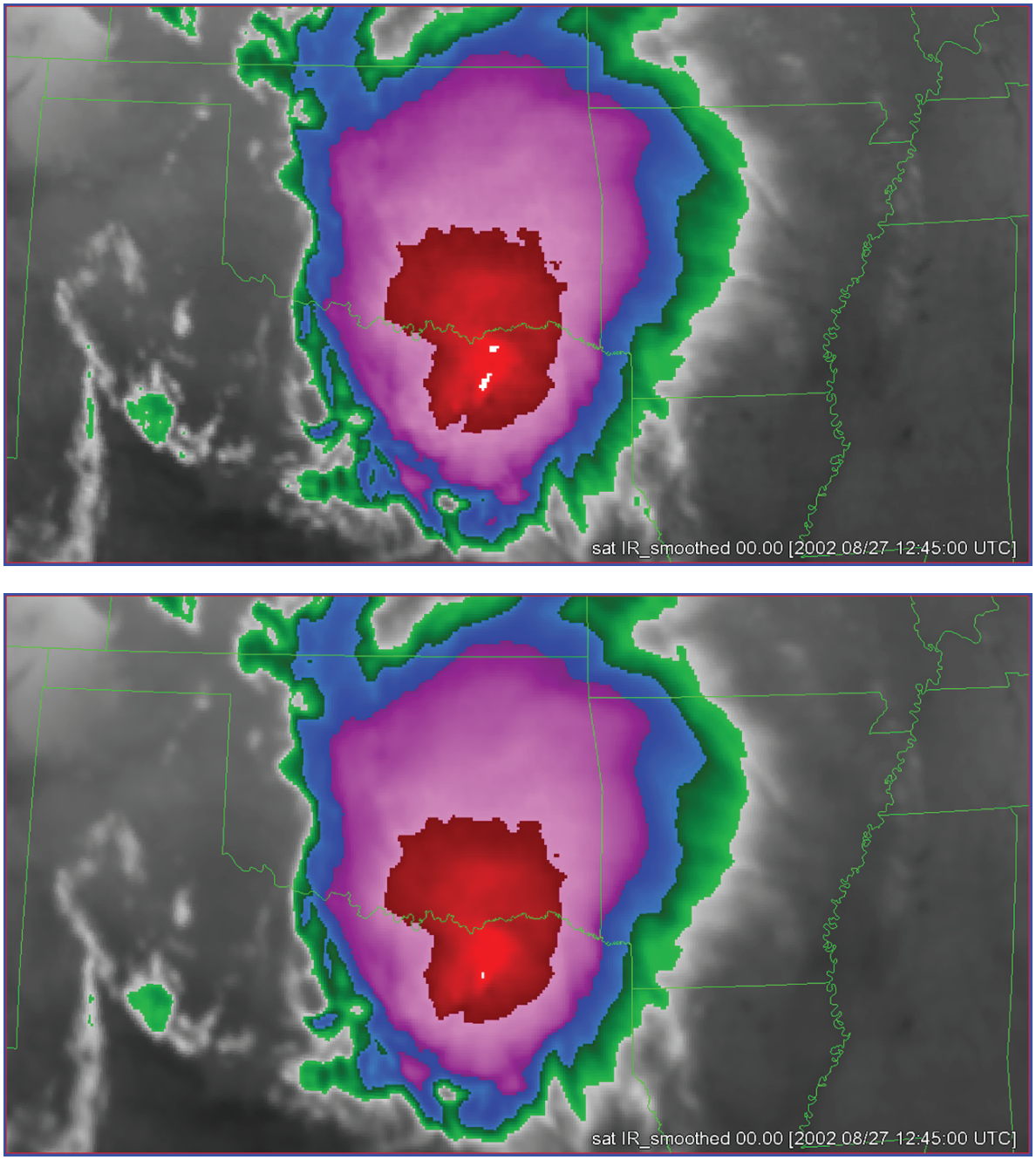


Fig. 16.4 Top: The infrared image of Fig. 16.3a smoothed with a box kernel with a half-size of 3. Bottom: The same infrared image smoothed with a Gaussian kernel with a sigma of 3 (and half-size of 9)

the result wherever there are high values in the input image, one needs to normalize the kernel result at a pixel by the smoothed value at that pixel. The kernel used in Fig. 16.5 returns large magnitudes for thin vertical lines, because if one were to think off it as a topographic map, the weight matrix is appears like a ridge. Of course, as Fig. 16.5 shows, the matched filter is not perfect – the operation returns large values for anything where the values of the left are higher than the values on the right, not just thin lines.

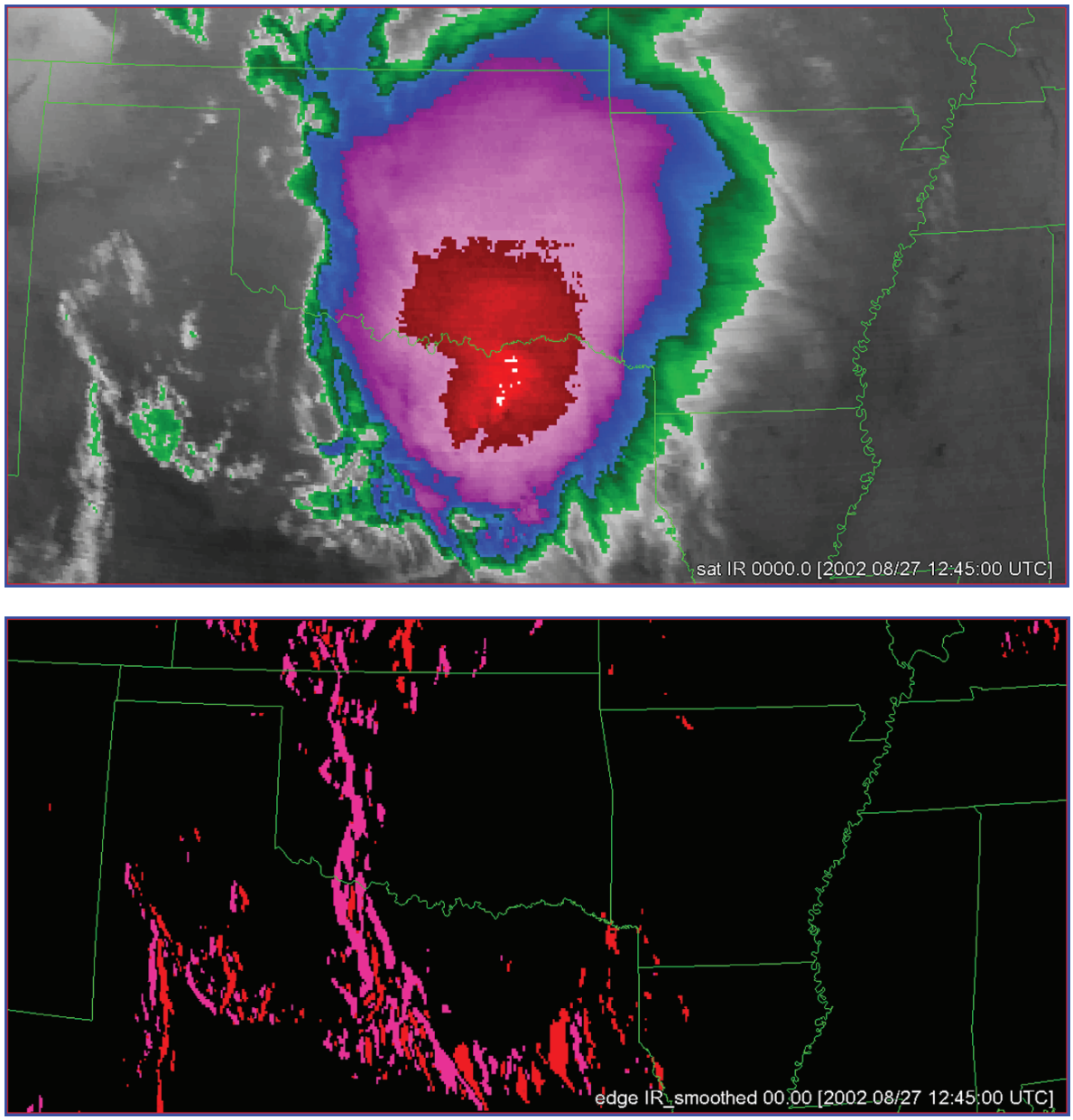


Fig. 16.5 Bottom: the effect of applying a vertical-line detection kernel to the infrared image on the top

16.4.3 Filter Banks

The matched filter idea may be used to find objects in an image. Unfortunately, though, convolution filters are both scale and orientation dependent. Thus, if one needs to find ridges that are five pixels thick, the convolution kernel needs to be five pixels thick. If the boundary of interest is not vertical, but horizontal, the matched filter needs to have its ones and zeros oriented horizontally. This makes matched filters extremely hard to use to find objects whose scale and orientation are not known in advance.

Most commonly, matched filters to find objects are used as part of a filter bank, as shown in Fig. 16.6. Several filters of different sizes and orientations are used to filter an image. The final result at a pixel is determined

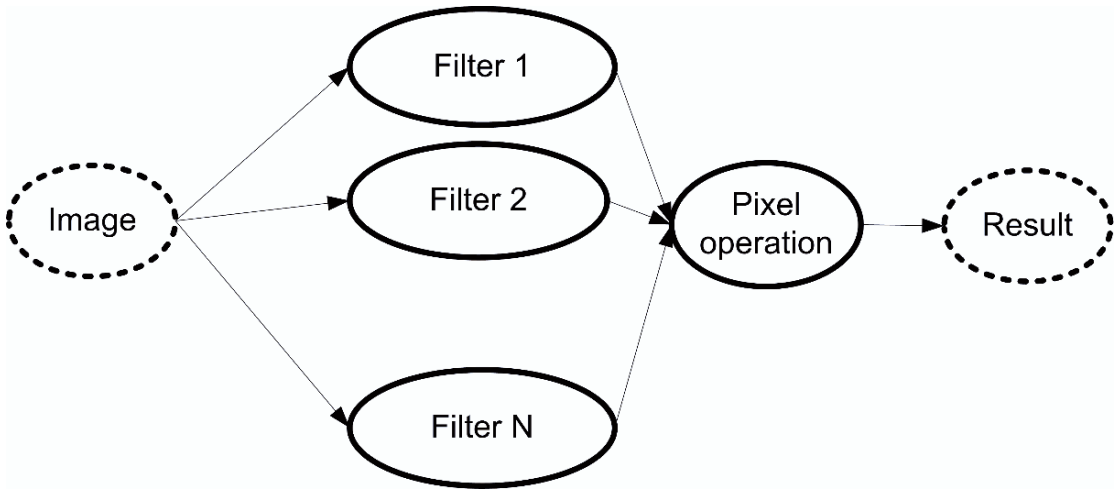


Fig. 16.6 A matched filter is typically used as part of a filter bank, with filters of different scales and operations. The best response from the individual filters is chosen at each pixel to provide the final result

by combining the result of the different convolution filters at that pixel. The combination method may be to take the average of the individual filter responses or to choose the best response, for example the maximum.

A general purpose smoothing convolution filter that can be used to match regions of different sizes and orientations is shown below:

$$W_{xy} = \frac{1}{{\sigma_x}^2 + \sigma_y^2} e^{\frac{(x\cos\theta)^2 + (y\sin\theta)^2}{-({\sigma_x}^2 + \sigma_y^2)}}$$

This is a Gaussian, where the two sigmas determine the vertical and horizontal scales while theta represents the orientation off the object that could be matched.

A number of boxcar convolution filters at different orientations (but not scales) are used in a filter bank to identify storm fronts by Wolfson et al. 1999. They then choose the maximum filter response to be the final output of the smoothing operation. Doing so achieves the nice effect of smoothing along the storm front.

There are several drawbacks to performing multi-scale or orientation analysis using a matched filter bank. Because repeat convolutions have to be performed on the original image, filtering can take a very long time. Also, because the results of the filters are not related, it may not be possible to perform higher level operations based on just filter banks. Several simplifications are possible in order to improve the efficiency and usability of filter banks. Firstly, as Section 16.3.5 illustrates, convolution itself may be sped up to by taking advantage of Fourier transforms and separable filters. If the resulting features are related, it is possible to first identified the objects, and then combine them outside the filter bank. If the resulting images themselves are related, it is possible to use wavelets to perform higher level operations on the related set of images.

16.4.4 Missing Data and Image Boundaries

Often, parts of the domain will not have been measured. The sensor may have had equipment problems. The point of view of the sensor may have been such that some part of the domain is out of range. The beam may have been blocked. Yet, numerical operations like image processing operators can not deal properly with such missing data. If one needs to compute a local average around the center pixel:

5	2	X
4	3	6
4	2	X

and "X" denotes a pixel that was not measured, how can a local average be computed? There are two broad approaches: (a) compute the average only on the non-missing pixels, in which case, the answer would be $26/6 = 4.3$ or (b) assume that a pixel when missing

is a "typical" value, say 5, in which case, the answer would be $36/8 = 4.5$.

Computing the value using only non-missing data usually yields better, more representative results with fewer artifacts. However, the second approach of filling unmeasured pixels with a default value is conducive to several optimizations, in particular of performing operations in a transform domain. The problem, of course, is of correctly choosing the default or "typical" value.

The same problem occurs in a different guise when the operations lead to the edge of the image. The simplest approach is to add an imaginary row or column, assuming that the entire row/column is missing and use one of the above two approaches. The exception is in the case of radar data where the radials "wrap" back around, so that the boundary condition is only when the data go out of radar range.

16.4.5 Speeding Up Convolution Operations

One problem with using Gaussian filters is that the scale of the filters tends to get quite large. This translates directly to filter size. So, the larger the filter, the more time it takes to compute a local average. If Gaussian filters are used in a filter bank, this loss of performance adds up to become a critical bottleneck. There are two ways to speed up a matched filter bank made up of Gaussian filters.

Convolution can be performed in Fourier transform space. The original image and awaiting matrix are both transformed into Fourier space. In Fourier transform space, convolution is merely pixel to pixel multiplication. Thus, convolution even with very large weighting matrixes takes only as long as the time it takes to compute the two Fourier transforms – the larger the kernel the more dramatic the speed up (Lakshmanan 2000). There is one drawback, however, to performing convolution in Fourier transform space. In order to compute a Fourier transform, it is necessary for the entire image to consist of valid values. If certain pixels within the grid went unsensed, Fourier transform methods cannot be used directly. Instead, in the preprocessing step, missing pixels have to be set to some default value. Traditionally, this default value is either zero or the mean of the entire image.

The following steps will perform convolution in Fourier transform space:

1. Pad weighting kernel to nearest power of 2 or other small prime
2. Compute Fourier Transform of weighting kernel
3. Pad image to nearest power of 2 or other small prime
4. Preprocess the image and fill all missing pixels with a default value, say zero
5. Compute Fourier Transform of image
6. Multiply the two transforms pixel-by-pixel
7. Compute the inverse transform of image

The reason for padding the image for computing the Fourier transform is that fast methods (called Fast Fourier Transforms, FFTs) exist to compute Fourier transforms on digital data.

Fourier transform methods cannot be used on images, such as radar data, that commonly have missing values. A second alternative exists for speeding up convolution operations. This is to formulate the weighting matrix as a separable function. For example, the Gaussian kernel without any orientation can be written in separable form as:

$$W_{xy} = \frac{1}{\sigma^2} e^{\frac{(x)^2+(y)^2}{-2\sigma_x^2}} = \frac{1}{\sigma^2} e^{\frac{(x)^2}{-2\sigma_x^2}} e^{\frac{(y)^2}{-2\sigma_x^2}}$$

Separable convolution filters can be implemented in a fast manner by first processing the image row by row, and then processing it column by column. If the weighting kernel were 25×25, convolution using the inseparable form would require 625 operations at every pixel. On the other hand, the separable form can get away with just 50. Therefore, the separable form of convolution can lead to significant speedups, although not as dramatic as the Fourier transform methods. For an approximation to oriented Gaussian filters in a separable form please see Lakshmanan 2004.

16.4.6 Wavelets

Wavelets are a multiresolution technique. They provide a way by which images may be broken up into sub-images such that one of the sub-images is a smaller but faithful representation of the larger image. The smooth images in Fig. 16.7 (S0, S1, S2, etc.) are decomposed to yield three detailed images (D0, D1, D3, etc.). That sub-image can itself be broken up

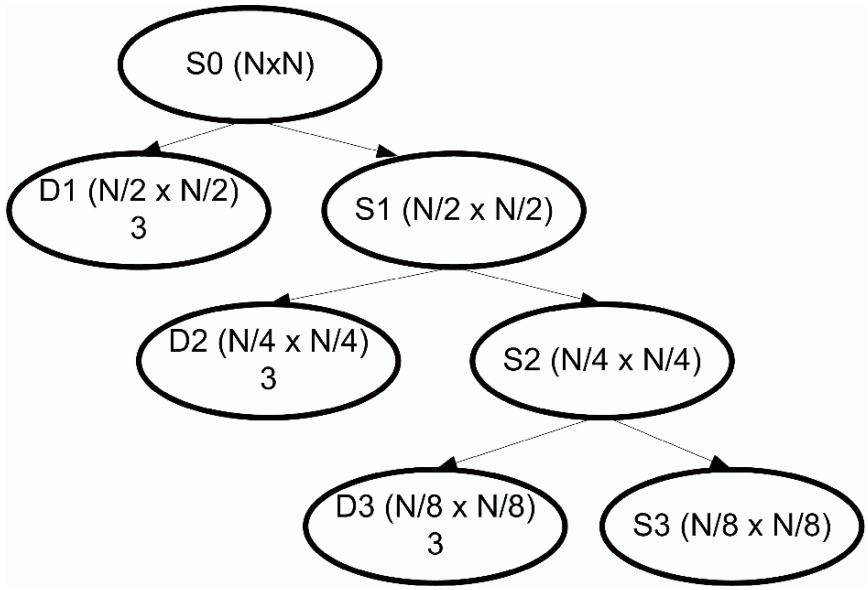


Fig. 16.7 Using wavelet analysis, images are decomposed into detailed images (left) and a smaller smoothed image (right)

into the more sub images. At each stage, the four sub images can be combined to yield the larger image that was decomposed to yield the sub images.

In order to decompose images such that the smoothed images have the above relationship, only specific convolution weight matrices may be used. The most commonly used convolution filters in wavelet analysis are the Haar function and the Daubechies p-functions. If all that is required is to be able to process images at multiple scales, wavelets are overkill. Simply filtering the image using Gaussian filters at different sigmas may suffice.

16.4.7 Edge Finding

The line finding filter described in Section 9.3.2 is not a robust way to find edges. Just as using a boxcar kernel to smooth images leads to abrupt transitions, using a box-like line finding filter results in smudging of the lines that are detected. To identify lines in an image, use the Canny edge-finding filter (Canny 1986). The Canny filter involves using the weighting matrix shown below:

$$W_{xy} = \left(\frac{1}{2} - \frac{x^2 + y^2}{\sigma^2}\right) e^{-\frac{x^2+y^2}{\sigma^2}}$$

After applying the above convolution filter (known as the Laplacian of a Gaussian because it is obtained by differentiating the Gaussian equation twice), look for zero crossings in the convolution result. Connecting up the zero crossings provides the location of the edges in the image.

16.4.8 Texture Operations

Convolution involves computing the weighted average of the pixel in the neighborhood of a central pixel. Other operations, however, can be performed once the values of the pixel's neighbors have been extracted. In fact, the boxcar kernel may be thought off as a statistical operation – the mean – on the neighboring values. Taken together, these statistical operators are called texture operators.

Texture provides a good way to distinguish different objects based on something other than just their data values. Even if two objects have the same mean value, the distribution of values within the objects may be different. Texture operators attempt to capture this difference in the distribution of data values. Commonly used texture operators include those based on second and third order statistics. Variance and standard deviation capture the range of values – high values of these are associated with "noisy" regions. Homogeneity is a third order statistic that captures how smooth the data values are in the neighborhood of the pixel. Another very useful texture operator is the entropy, which is

Shannon's measure of information content. It is computed by taking all the data values in the neighborhood and forming a histogram. If p is the fraction of pixels in each bin of the histogram, the entropy is given by:

$$\sum_i p_i \log(p_i)$$

The entropy is low in regions where all the pixels fall in the same bin (because the logarithm of one is zero and the probability in the all other bins is zero as well). The entropy is highest in regions where there is a large diversity of pixel values. Typically, very high entropies are associated with noise while very low entropies are associated with instrument artifacts. Of course, the actual thresholds have to be found by experiment.

16.4.9 Morphological Operations

In addition to convolution filters and texture operators, the neighborhood values may be sorted and operators

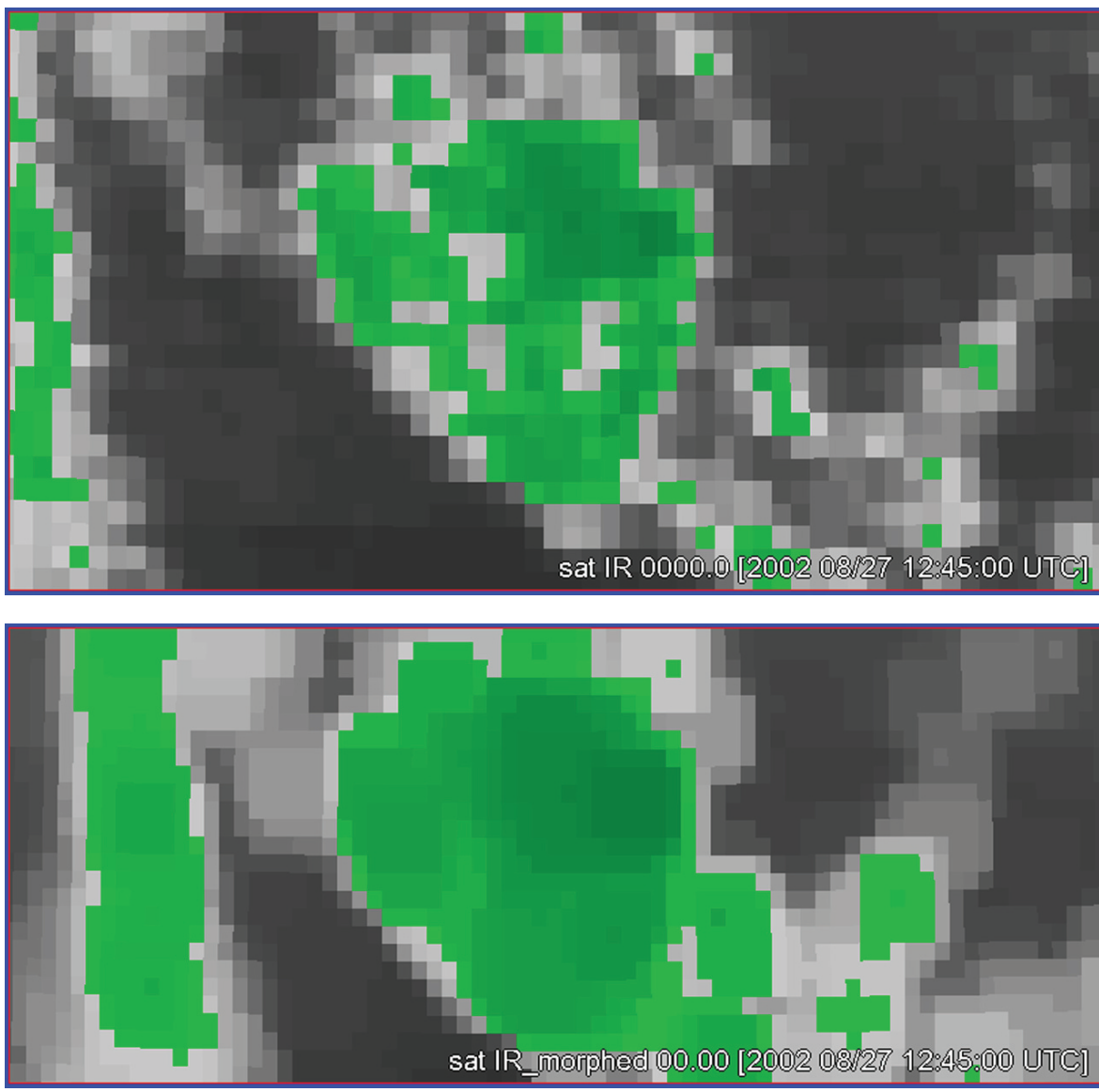


Fig. 16.8 Spatial dilation and erosion operations may be performed by replacing a pixel with the maximum or minimum of the values of its neighbors

Fig. 16.8 (*Continued.*)

based on the sorted values may be applied to the images. The median value off the neighborhood values is an excellent smoothing filter. The median filter works especially well, and should be chosen in preference over Gaussian filters, when the noise has speckle characteristics.

Taking the minimum value of the pixel values in a neighborhood has the effect of eroding the image spatially, so that regions become smaller and small regions get removed. Similarly, taking the maximum has the effect of dilating the image spatially. The minimum and maximum are affected detrimentally, if there is noise present in the images. Taking something like the fifth or 95th percentile may help tradeoff the noise characteristics of the image. Alternately, taking the second-lowest or the second-highest value in the neighborhood also helps to reduce the impact of noise. Spatial erosion and dilation where the second lowest and highest values are chosen is shown in Fig. 16.8.

16.4.10 Filter Banks and Neural Networks

So far, we have looked at the number of operations that can be performed on the neighborhood values of the pixel. These include convolution filters (boxcar, Gaussian, matched filters, orientation detectors), texture operators (variance, homogeneity, entropy) and morphological operators (erosion, dilation). Several of these filters may be applied in to an image, either in parallel or one after the other, to an image as part of a filter bank as shown in Fig. 16.9.

For every pixel in the original image, we will obtain a vector of filter results. A neural network or some other classifier can take all of these input and classifying each vector into two or more categories. In other words, each pixel provides a pattern to the neural network. Since a $1{,}000 \times 1{,}000$ image will provide one million patterns, one may have to be selective about whether all the pixels in an image get presented to the neural network for training and/or classification. Because many of the pixels in an image are highly correlated, one must be careful to not assume that the patterns from a single image are independent training samples to the neural network. A good rule of thumb is to treat all the patterns from a single image as simply one data case when deciding whether one has enough of a training or validation set.

As an example of an AI application that proceeds from images to filters through a neural network towards the classification result, consider the radar reflectivity quantity control system described in (Lakshmanan 2006). In that application, radar reflectivity, velocity and the spectrum width polar data are

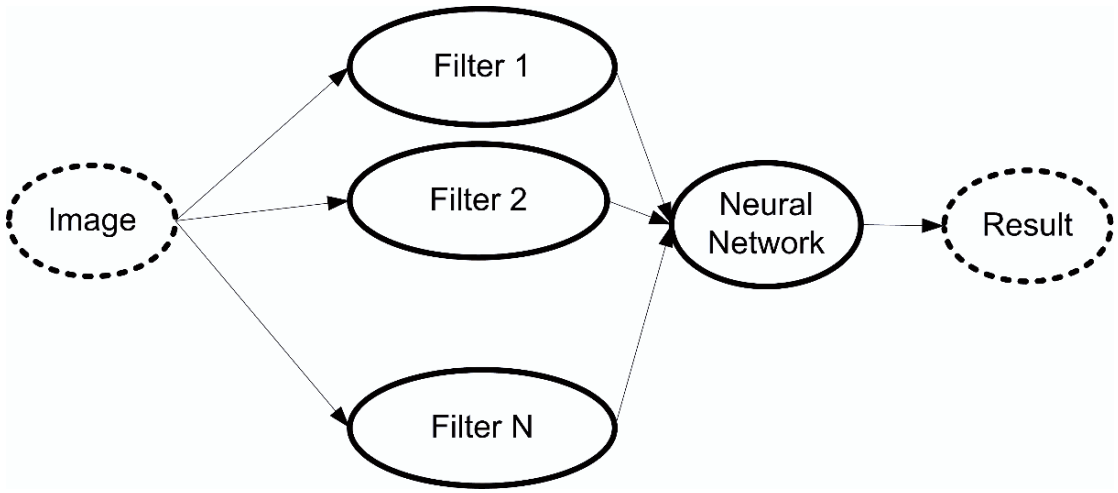


Fig. 16.9 The output of a filter bank may be used to provide patterns to a neural network for classifying an image pixel-by-pixel

converted to a uniform resolution and indexed polar grid. A variety of texture operators are applied to the three polar grids at each elevation. The results of the filters at each pixel form the patterns that are presented to a classification neural network. The output of the neural network is thresholded at 0.5 to determine whether the pixel in question corresponds to good data or to non-meteorological artifacts. Not all the pixels are presented to the neural network. Instead, a preclassification step classifies those pixels that can be done quite easily based on a rule engine. The pixels that are presented to the neural network comprise the hard to classify pixels. If the pixels are classified solely by the neural network and the rules engine, the resulting field will have different classifications even for adjacent pixels that are part of the same storm or artifact. In order that regions off the radar image all get classified the same, it is necessary to average the results of the classification over objects found in the images rather than treat the pixels as being independent. How to do this is the focus of the next section.

16.5 Segmentation: Images to Objects

So far, we have looked at ways of processing images. Convolution filters, texture operators, edge detection and morphological operators all have, as their output, images. In most applications, however, what is desired is to be able to identify objects and to classify or track these objects as entities. Classifying pixels makes no sense, because pixels are ultimately just an artifact of the remote sensing instrument. In this section, we will look at how to identify objects from grids.

16.5.1 Hysterisis, to Convert Digital Data to Binary Images

The process of identifying self-similar groups of pixels and combining them into objects is called segmentation. Segmentation algorithms work only on binary data. Therefore, the pixel values have to be converted to zeros and ones before segmentation can begin. Using just one threshold to convert the data into binary typically results in lots of very tiny objects (if the threshold is too high), a few very large objects (if the threshold is too low) or in objects that have lots of holes (if a moderate threshold is chosen).

To avoid the problems associated using only one threshold, employ hysteresis to convert pixel values into zeros and ones. Hysteresis involves picking two thresholds, say t1 and t2. Pixel values below the first threshold are always set to zero and pixel values above the second threshold are always set to one. Pixels with values between the two thresholds are set to one only if they are contiguous to a pixel that has already been set to one – this condition is easily applied during region growing, discussed in the section. The use of this technique mitigates, but does not completely prevent, the

problems associated to with a single threshold – the choice of the two thresholds t1 and t2 has to be made through experiment.

16.5.2 Region Growing

The segmentation process is done through region growing. In the region growing algorithm, contiguous pixels are assigned the same label such that all the pixels that have the same label together form a region. The steps of a region growing algorithm are listed below:

1. Initialize an image called the label image. All of its pixels are set to zero.
2. Initialize current label to zero.
3. Walk through the image pixel by pixel and check if the pixel value of the image is greater than t2 (see Section 16.4.1 on hysteresis) and if the label at this pixel is zero. If both conditions are true:
   (a) Increment current label.
   (b) Set label at this pixel to be the current label.
   (c) Check all eight neighbors of this pixel. If a neighbor's pixel value is greater than t1, repeat steps b and c at the neighboring pixel.

At the end of the above steps, the label image has a region number assigned to every pixel. Pixels where the label image has a value of zero do not belong to any region. Two pixels with the same label are part of the same region. Therefore, region properties may be computed from the original image's pixel values by maintaining a list of statistics (one for each region), walking through the pixels of the original image and updating the statistic for the corresponding region (read out from the label image) that the pixel belongs to. This way, a vector of properties or statistical features is obtained for each region. These features may be presented to a neural network or other classifier to classify the identified regions into two or more categories.

16.5.3 Vector and Hierarchical Segmentation

Although with hysteresis, one uses two thresholds to reduce the incidence of disconnected regions, the determination is made ultimately on a single pixel value. The incidence of disconnected regions could be reduced even further if the values of neighboring pixels could be considered when creating the regions. In other words, it would be better if a filter bank could be applied to an image, and the results from all the filters in the filter bank could be used to determine which region a pixel belongs to. Such a segmentation technique is called a vector segmentation technique because it operates on a vector of inputs, not just one value at every pixel.

Another drawback of using the hysteresis-based region growing approach is that it is not possible to obtain hierarchical regions with overlapping thresholds. In real-world AI applications, it helps to be able to identify regions and place them into larger regions. Regions, sorted by size, can be used for different tasks and to evaluate different types of constraints. Such a segmentation technique is termed a hierarchical approach.

The classical hierarchical segmentation technique is the watershed algorithm of (Vincent and Soille 1991). The technique consists of first sorting the pixel values within the image in ascending order of magnitude and then slowly raising a "flood" level. Connected regions, identified through region growing, form a hierarchy with the flood level determining how high up in a hierarchy a region exists before it is subsumed into a larger region. The saliency of a region, the maximum "depth" of a region, provides an indication of how important it is. Unfortunately, watershed segmentation works very poorly in images with statistical noise. Watershed segmentation may work quite well on model fields and other smooth datasets, so it is worth trying before attempting more complex techniques. Statistical noise is a given in most remotely sensed images, so a better hierarchical technique, ideally one that works on vector data, is required.

Lakshmanan 2003 describes a technique where an explicit trade off is made between the self-similarity of a region and the idea that a region should be compact and consist of contiguous pixels. A pixel is similar to a region if the Euclidean distance between the result of a filter bank applied to the image at that pixel is close to the mean of the filter bank results of all the pixels that are already part of the image. This is essentially the K-Means clustering approach, of arbitrarily choosing K regions and then updating the regions based on which

pixels get added or removed from them. When the set of pixels in a region does not change, or if the magnitude of those changes is quite small, the clustering is stopped. This vector segmentation approach is made hierarchical by using steadily relaxing the allowed inter-cluster distance before regions are merged.

16.6 Processing Image Sequences

So far, we have considered processing images (or spatial grids) individually. In many applications, the temporal change of spatial data is important. For example, one might wish to study the change in forest cover over a decade or the movement of a hurricane over ocean. In motion picture processing, individual images are termed "frames" and a set of frames arranged in temporal order forms a "sequence". There has been quite bit of research into techniques for processing image sequences, mainly for applications such as the compression of video and for automated security monitoring using video cameras.

16.6.1 Detecting Change

The most common requirement in processing image sequences is to simply determine whether a change has occurred. The simplest way of identifying changes is to compute a pixel-by-pixel difference between selected frames of a sequence. Pixels where the absolute maximum of the change is high indicate locations where a change has occurred.

The differencing technique works well for sequences where objects suddenly appear or disappear. For example, in a forestry application, it is possible to use differencing to monitor whether areas of a forest have been cleared.

The differencing technique works poorly in applications where the objects are moving from one location to another. This is because the magnitude of the difference field will be high only in areas where the object has completely moved to or away. Where there is an overlap, the magnitude of the difference will be small. Thus, a single movement might appear to be two separate areas of change. This limits the utility of differencing techniques in applications such as storm tracking.

16.6.2 Tracking by Object Association

One way to mitigate the problems associated with pixel-by-pixel differencing is to compute differences on a region-by-region basis. In other words, segment the frames of a sequence, associate objects between frames and then tabulate changes in aggregate statistics (such as size, mean value, etc.) of the regions.

This is easier said than done. Several hard problems arise in the technique outlined above. Segmentation is a notoriously noisy operation – the change of magnitude in a few pixels can drastically change the objects that are identified. Since successive frames in a sequence are slightly different, the results of segmentation on these frames may result in dramatically different region identification. This makes associating regions problematic. Even if the segmentation problems are resolved satisfactorily so that slightly different images yield only slightly different regions, the region association problem is not insignificant. In storm tracking, for example, individual storm cells may split or merge – this needs to be accounted for in the object association.

If the objects are large enough and move slow enough for them to overlap significantly, the association problem is not difficult. A minimum overlap in terms of spatial correlation may suffice for associating objects. This is the strategy commonly employed in tracking mesoscale convective systems. See for example Carvalho and Jones 2001.

When there are numerous small features identified, it can be unclear as to what the optimal assignment ought to be. A principled way of associating objects would be to minimize a global measure of fitness, such as the total Euclidean distance between the vectors of properties between associated objects. This is the approach followed by Dixon 1993, where a linear programming approach is employed to minimize a least squares metric. A less principled approach, but one that works quite well, is to extrapolate the movement of regions and then simply choose the region in the next frame closest to the anticipated location. This is the method employed by Johnson et al. 1998.

16.6.3 Estimating Movement Using Optical Flow

Object association techniques tend to identify movement only at the scale of the objects. If the objects are ephemeral or difficult to identify, the quality of the movement estimated from object tracking methods suffers.

A better way of estimating movement may be to ignore object identification altogether. Instead, the image is considered a fluid field and the movement at a pixel is assumed to be that movement which when removed from the current field would result in the smallest magnitude difference field to the previous frame. To compute this, a neighborhood of pixels around a central pixel is moved around in the previous frame until the point at which the difference is minimal is identified. This movement is the movement at that pixel. This process is repeated for each of the pixels in the current frame. Such a technique usually minimizes the sum of the squares of the differences in pixel values, and is therefore termed spatial cross-correlation. Tuttle and Gall (1999), for example, take rectangular subgrids of radar data, move it around the previous frame and use the minimal movement to estimate movement of tropical cyclones.

Wolfson et al. 1999 improved the basic cross-correlation technique by smoothing the images with a filter bank consisting of elliptical convolution filters (to smooth along the storm front), added smoothing criteria so that adjacent pixels do not have wildly different motion estimates and incorporated a global motion vector to eliminate outliers at the expense of being unable to track circular movements such as hurricanes. The Lagrangian technique of Turner et al. 2004 explicitly formulates this as a multi-variable optimization problem but avoids the global motion vector criterion to retain the ability to track continent-scale flows in multiple directions.

Object association techniques yield more accurate wind speeds for small-scale storms, but the movement of larger-scale features is better predicted by optical flow methods. One drawback of optical flow methods is an inability to obtain long-term statistics about the trends of object properties, because no objects are identified in these techniques. Lakshmanan et al. 2003 introduced a hybrid technique that estimates movement of objects identified through hierarchical segmentation using optical flow techniques. Because the optical flow movement is estimated, not on rectangular subgrids of the image, but on templates the size and shape of the identified clusters, the clusters can be displaced through multiple historical frames to yield object statistics.

16.6.4 Trends of Geographic Information

So far, we have looked at temporal changes in objects, such as storm systems, that move. How about obtaining the temporal properties from a stationary viewpoint, such as the total precipitation that falls within the boundaries of a city?

The simplistic approach is to map the geographic data onto the same grid as the remotely sensed data and then perform the accumulation operation on the pixels that correspond to the city. However, gridding geographic information system (GIS) data is not always practical. Ideally, it would be possible to leave GIS data in its original vector form and compute spatial properties that fall within the vector.

To decide whether a pixel falls within a polygon, simply map the vertices of the polygon to the gridded

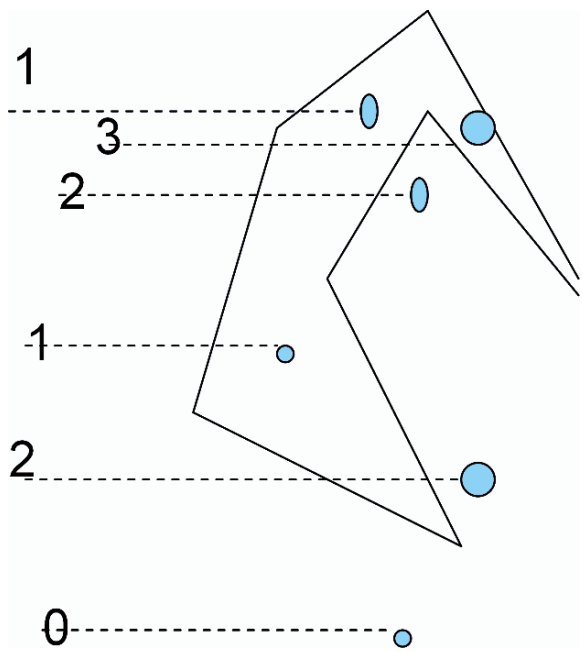


Fig. 16.10 Count the number of times an edge of a polygon is encountered to determine whether a pixel in question falls within the polygon

reference i.e. convert the polygon vertices to (x,y) locations. Assuming, for simplicity that the polygon is completely within the grid, walk row-wise from the grid boundary to the pixel of interest, counting the number of times that the sides of the polygon are crossed (this can be done by finding the intersection point of the line corresponding to the row and the line corresponding to each of the edges of the polygon – see any geometry book for details on the formulae to do this). If the total number of crossings is odd both when walking row-wise (see Fig. 16.10) and when walking column-wise, the pixel falls inside the polygon and its pixel value contributes to polygon statistics.

16.7 Summary

In this chapter, we have looked at how image processing plays a key role in artificial applications that operate on spatial data. We discussed techniques to convert in situ observations into spatial grids. We also examined ways to reduce noise, find edges and identify objects from spatial grids. Finally, we looked at purely image-driven AI applications such as tracking and the extraction of spatial properties within vector boundaries.

References

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *8*(6), 679–698.

Carvalho, L., & Jones, C. (2001). A satellite method to identify structural properties of mesoscale convective systems based on the Maximum Spatial Correlation Tracking Technique (MASCOTTE). *Journal of Applied Meteorology*, *40*(10), 1683–1701.

Cressman, G. P. (1959). An operational objective analysis system. *Monthly Weather Review*, *87*, 367–374.

Dixon, M., & Wiener, G. (1993). TITAN: Thunderstorm identification, tracking, analysis and nowcasting: A radar-based methodology. *Journal of Atmospheric and Oceanic Technology*, *10*, 6.

Johnson, J. T., MacKeen, P. L., Witt, A., Mitchell, E. D., Stumpf, G. J., Eilts, M. D., et al. (1998). The Storm Cell Identification and Tracking (SCIT) algorithm: An enhanced WSR-88D algorithm. *Weather and Forecasting*, *13*, 263–276.

Koch, S. E., DesJardins, M., & Kocin, P. J. (1983). An interactive Barnes objective map analysis scheme for use with satellite and conventional data. *Journal of Climate and Applied Meteorology*, *22*, 1487–1503.

Lakshmanan, V. (2000). Speeding up a large scale filter. *Journal of Oceanic and Atmospheric Technology*, *17*, 468–473.

Lakshmanan, V. (2004). A separable filter for directional smoothing. *IEEE Geoscience and Remote Sensing Letters*, *1*, 192–195.

Lakshmanan, V., Rabin, R., & DeBrunner, V. (2003). Multiscale storm identification and forecast. *Journal of Atmospheric Research*, 367–380.

Lakshmanan, V., Fritz, A., Smith, T., Hondl, K., & Stumpf, G. J. (2007). An automated technique to quality control radar reflectivity data. *Journal of Applied Meteorology*, *46*, 288–305.

Oliver, M. A., & Webster, R. (1990). Kriging: A method of interpolation for geographical information system, Int'l. *Journal of Geographical Information Systems*, *4*(3), 313–332.

Turner, B., Zawadzki, I., & Germann, U. (2004). Predictability of precipitation from continental radar images. Part III: Operational nowcasting implementation (MAPLE). *Journal of Applied Meteorology*, *43*(2), 231–248.

Tuttle, J., & Gall, R. (1999). A single-radar technique for estimating the winds in tropical cyclones. *Bulletin of the American Meteorological Society*, 80, 683–685.

Vincent, L., & Soille, P. (1991). Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *PAMI(13)*, *6*, 583–598.

Wolfson, M., Forman, B. E., Hallowell, R. G., & Moore, M. P. (1999). The growth and decay storm tracker. *Eighth Conference on Aviation* (pp. 58–62). Dallas, TX: American Meteorological Society.

Fuzzy Logic Applications

17

John K. Williams, Cathy Kessinger, Jennifer Abernethy, and Scott Ellis

17.1 Introduction

Fuzzy logic originated in the mid-1960s with the work of Lotfi Zadeh, as described in Chapter 6. However, it wasn't until the 1990s that it was widely recognized as a valuable tool in the atmospheric and other environmental sciences. One of fuzzy logic's first successful applications in the atmospheric sciences was the Machine Intelligence Gust Front Detection Algorithm (MIGFA) developed at the Massachusetts Institute of Technology's Lincoln Laboratory (Delanoy and Troxel 1993). Since then, a wide range of environmental science problems have been successfully addressed using fuzzy data analysis and algorithm development techniques. The purpose of this chapter is to supplement the introduction to fuzzy logic presented in Chapter 6 by describing a few selected applications of fuzzy logic that provide a flavor of the power and flexibility of this approach.

John K. Williams (✉), Cathy Kessinger and Jennifer Abernethy
Research Applications Laboratory, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307, USA;
Email: jkwillia@ucar.edu, kessinge@ucar.edu, aberneth@ucar.edu
Phone: 303-497-2822, 303-497-8481, 303-497-2871;
Fax: 303-497-8401

Cathy Kessinger and Scott Ellis
Earth Observing Laboratory, National Center for Atmospheric Research, P.O. Box 3000, Boulder, CO 80307, USA;
Email: kessinge@ucar.edu, sellis@ucar.edu;
Phone: 303-497-8481, 303-497-2076; Fax: 303-497-8770

Jennifer Abernethy
Department of Computer Science, University of Colorado at Boulder, 430 UCB, Boulder, CO 80309, USA;
Email: aberneth@ucar.edu; Phone: 303-497-2871;
Fax: 303-497-8401

The unique contribution of fuzzy logic is that it provides a practical approach to automating complex data analysis, data fusion, and inference processes that are usually performed by human experts with years of formal training and extensive experience. For instance, in developing a weather forecast, meteorologists consult data from various observations and models, each having a different level of relevance and reliability. Telltale patterns are sought in satellite and radar images, with contamination identified and disregarded. Soundings and surface data are analyzed and compared for consistency. Numerical weather models are consulted and weighed, with attention given to their past performance. Each bit of information provides a new piece of the puzzle, and may suggest a reanalysis of other data in an iterative process. Eventually, the available information is synthesized to form an overall consensus view, and perhaps also an assessment of confidence in the resulting forecast. This is exactly the type of complex procedure that a fuzzy logic expert system might automate. A fuzzy logic algorithm solution would first develop modules for analyzing and performing quality control on the various sources of information. For example, standard image processing techniques might be used to measure local characteristics in the radar or satellite data, or to identify features such as fronts by convolving appropriate templates with the image. Physical models or function approximation, including trained neural networks or other empirical models, might be used to relate raw sensor measurements to a quantity of interest – for example, determining temperature profiles from radiometer measurements. Statistical analyses might be employed to help determine data quality, or to determine conditional probabilities based on historical data. Human input such as hand-drawn

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

fronts or boundaries might also be incorporated. The evidence supplied by these multiple data sources would be weighted by quality and by relevance, and used to perform fuzzy inference. The inference steps might be multi-layered or even iterative, feeding back into the data analysis; they could be tuned based on a training set or dynamically adjusted online using recent verification data. As this description suggests, a fuzzy algorithm can be quite complex – enough so, in fact, to represent many aspects of a human expert's reasoning process. Like human experts, the best fuzzy logic algorithms make use of all available information, saving any hard thresholding or binary decisions until the last step whenever possible. They also often provide a "confidence" value that lets downstream decision processes know how trustworthy the results are likely to be.

The applications of fuzzy logic presented in this chapter are drawn from those developed at the National Center for Atmospheric Research (NCAR) with which this chapter's authors are most familiar. The use of fuzzy logic at NCAR is extensive and can be divided into several problem areas: improving sensor measurements, performing data quality control, characterizing current weather phenomena, forecasting future weather, knowledge discovery and verification. Some of the most common application areas include decision support systems for aviation, road weather, national security, and societal impacts.

For example, one of the first applications of fuzzy logic at NCAR was the development of an automated algorithm for detecting microburst signatures in Doppler weather radar data (Albo 1994, 1996, building on work by Merritt 1991). Microbursts are small scale downdrafts usually associated with thunderstorms (Fujita and Byers 1977; Fujita and Caracena 1977; Hjelmfelt 1988) that can be extremely hazardous to aircraft when encountered at low altitudes. The Microburst Automated Detection (MAD) algorithm analyzes the radar data to look for areas of low-level, divergent outflow occurring near a storm, having a characteristic size and shape and a lifetime of a few minutes. If a microburst is identified, the sensitivity of detection in that region is automatically increased since microbursts often occur in lines or clusters. The automated microburst detection capability provided by radar and other ground-based warning systems, along with pilot training, has virtually eliminated aviation accidents due to microbursts at fully-equipped U.S. airports since the mid-1990s.

Other fuzzy logic algorithms developed at NCAR include techniques for identifying hydrometeor types (e.g., rain, ice, snow) using polarimetric radar measurements (Vivekanandan et al. 1999; Barthazy et al. 2001), a system for identifying and forecasting aircraft icing conditions (McDonough et al. 2004; Bernstein et al. 2005), a method for generating automated weather forecasts for cities (Gerding and Myers 2003), systems for merging observation and model data to produce short-term thunderstorm forecasts (Mueller et al. 2003; Roberts et al. 2006), techniques for forecasting ceiling and visibility for aviation users (Herzegh et al. 2004), and a method for performing automated quality control on time series data (Weekley et al. 2003). In addition, fuzzy logic is playing an increasingly important role in verifying forecast products through the use of "object oriented" techniques that award partial credit even when the time and position matches of a forecast with subsequent measurements are imperfect (e.g., Davis et al. 2006). Fuzzy clustering techniques and self-organizing maps are also being used to analyze large datasets and attempt to identify "weather regimes" that may lead to improved forecasts.

Because it is impossible to give in-depth expositions of all of these applications in a single chapter, detailed descriptions are provided below for just three: the NCAR Improved Moments Algorithm (NIMA) for improving Doppler wind profiler measurements, the Radar Echo Classifier (REC) technique for classification and quality control of Doppler radar data, and the Graphical Turbulence Guidance (GTG) system for forecasting clear-air turbulence. While these examples by no means supply a comprehensive catalog of how fuzzy logic may be applied in the environmental sciences, they do offer a suggestive overview of several successful applications of fuzzy logic concepts.

17.2 Improved Wind Measurements

Remote sensing of atmospheric quantities is a valuable capability for meteorological research and operational weather systems. For optimal performance of downstream applications, the input data must be of the highest possible quality with all suspect data recognized and identified as such. While trained researchers are often able to examine visualizations of raw or

processed sensor data and identify contamination or spurious measurement results while isolating the desired signal, it is increasingly worthwhile to automate these processes – particularly when the sensor data are needed for operational, real-time use. This section describes a fuzzy logic algorithm for extracting high-quality winds and wind confidences from velocity spectrum data such as those produced by wind profilers, radars, lidars, and other Doppler instruments. The method, known as the NCAR Improved Moments Algorithm (NIMA), uses a combination of image processing and fuzzy logic techniques and was deliberately constructed to mimic the reasoning of human experts. By weighing and combining various sources of "evidence" in a multi-step processing sequence, NIMA provides more robust and accurate measurements than those produced using traditional methods. NIMA was originally presented in Cornman et al. (1998) and Morse et al. (2002); a comprehensive verification study was given in Cohn et al. (2001), and NIMA's operational application to Doppler wind profiler data was described in Goodrich et al. (2002).

17.2.1 Doppler Moment Estimation

Although it was originally designed to operate on wind profiler data, NIMA may be applied to data from any of a number of Doppler instruments with appropriate modification. Such devices generally emit a focused, modulated signal of radio, light, or sound waves, detect backscattered echoes, and use signal processing techniques to analyze the resulting data and produce a velocity spectrum at a series of discrete distances called *range gates* along the beam. For instance, the National Weather Service's (NWS) Weather Surveillance Radar-1988 Doppler (WSR-88D; also called Next Generation Weather Radar or NEXRAD) radars currently have Doppler measurement range gates at 250 m intervals out to a slant range of 230 km, while the vertically-pointing wind profiler data presented below utilizes a 55 m range gate spacing up to about 2.5 km. The velocity spectrum represents the returned power for each radial velocity detected by the instrument (within its resolvable velocity range or *Nyquist interval*), and may be interpreted as showing the distribution of radial velocities for all the scatterers (hydrometeors, dust, aerosols, or variations in the air's refractive index) contained within a *measurement volume* around that location. The spectral power value at range gate r and velocity v may be represented as a function $f(r, v)$. In the absence of contamination or velocity aliasing (folding around the maximum resolved value or *Nyquist velocity*), the k^{th} moment of the spectrum for range r may be computed as

$$M_k(r) = \int_{-v_n}^{v_n} v^k \left[f(r, v) - N(r) \right] dv \qquad (17.1)$$

or the equivalent sum when the velocities v are discrete; here v_n represents the Nyquist velocity and $N(r)$ represents the random "noise" contribution to the signal at range r. The 0^{th} moment may be related to the total backscattering cross-section in the measurement volume, commonly called *reflectivity*; the 1^{st} moment gives the mean radial velocity; and the square root of the 2^{nd} moment, called the spectrum half-width, is related to turbulence and wind shear within the measurement volume. Together, these quantities – measured at each range gate along a beam and for a succession of times during which the instrument may be rotated or translated – provide valuable characterization of the state of the atmosphere. For a review of how Doppler weather radars and profilers work, the reader is referred to the comprehensive text by Doviak and Zrnić (1993).

17.2.2 The Spectrum Contamination Problem

Unfortunately, in practice, background radiation and random fluctuations in the signal detection, amplification, and other processing generate random "noise" that is added to each velocity spectrum value and must be accurately estimated in order for (17.1) to produce correct results. Moreover, the spectrum f is often contaminated by non-atmospheric returns such as clutter (caused when part of the signal bounces off the ground, ocean, or fixed or moving objects on those surfaces), point scatterers (e.g., birds, bats or airplanes), or radio-frequency interference (RFI) caused by radio sources like cellphones, microwave ovens, other radars or profilers, or the sun. One way that human experts detect such contaminants is by representing the spectral function $f(r, v)$ for each beam as a 2-D colorscaled image. For example, Fig. 17.1 shows data for a single beam

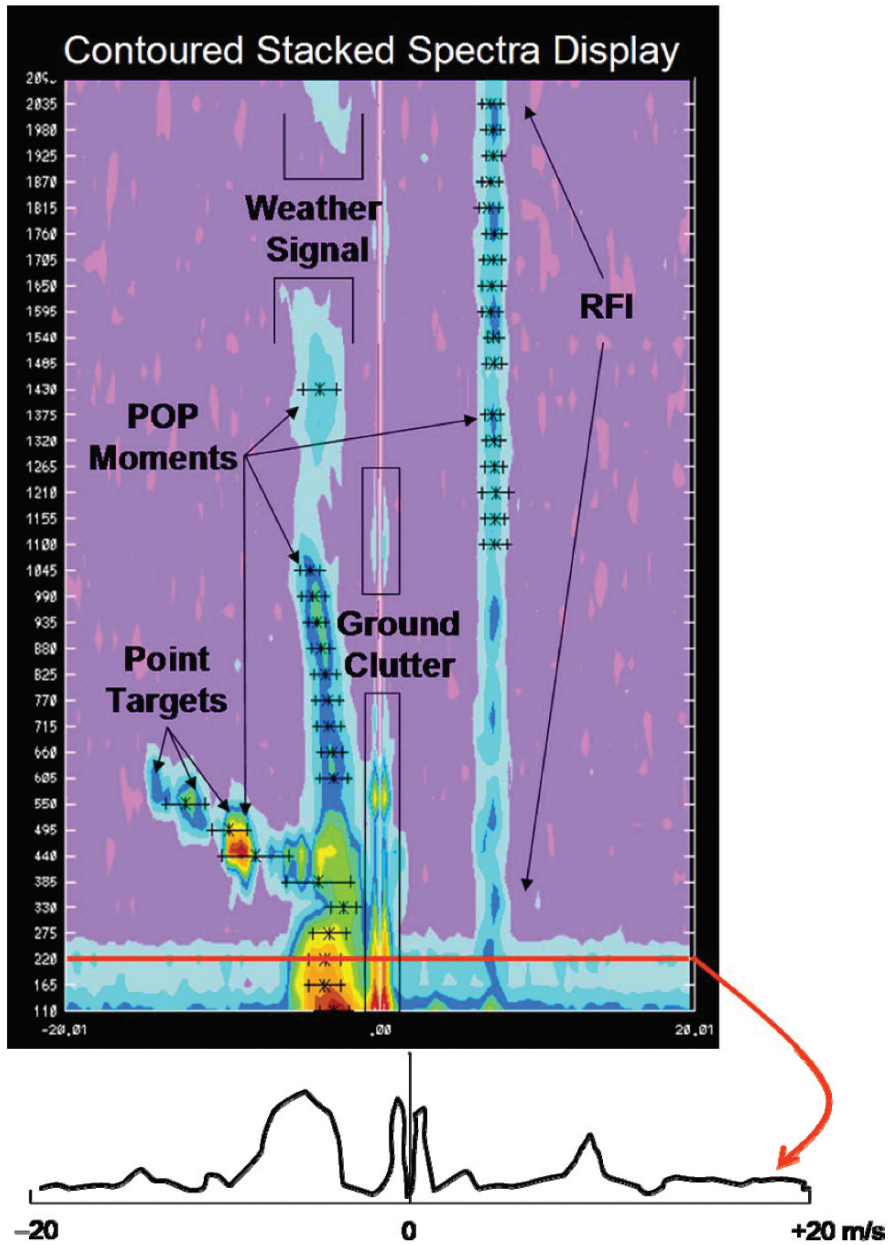


Fig. 17.1 (Top) Contoured, color-scaled stacked-spectra plot for a single beam from the Lemon Creek 915-MHz profiler in Juneau, Alaska, with range gates along the y-axis and radial velocity along the x-axis. Overlaid are the radial velocity ("*") and spectrum width ("+") from the traditional POP moment algorithm. "Features" comprising the weather signal and ground clutter, point target, and radio-frequency interference contamination are also indicated. (Bottom) The power spectrum represented as a line plot for the range gate at 220 m. Source: Adapted from Fig. 1 of Morse et al. (2002).

from the Lemon Creek 915-MHz profiler in Juneau, Alaska. Several "features" in the velocity spectra having continuity along both the range and radial velocity dimensions immediately become evident. Overlaid on the image are indications of the radial velocity ("*" symbols) and spectrum width ("+" symbols) estimates produced by a traditional processing method called Profiler On-line Program (POP; Carter et al., 1995). It is clear that POP is frequently being misled by the various sources of contamination and is therefore producing spurious moment estimates.

17.2.3 NIMA's Fuzzy Logic Solution

NIMA's goal is to mimic a human analysis of the spectral data by identifying the source of each feature and

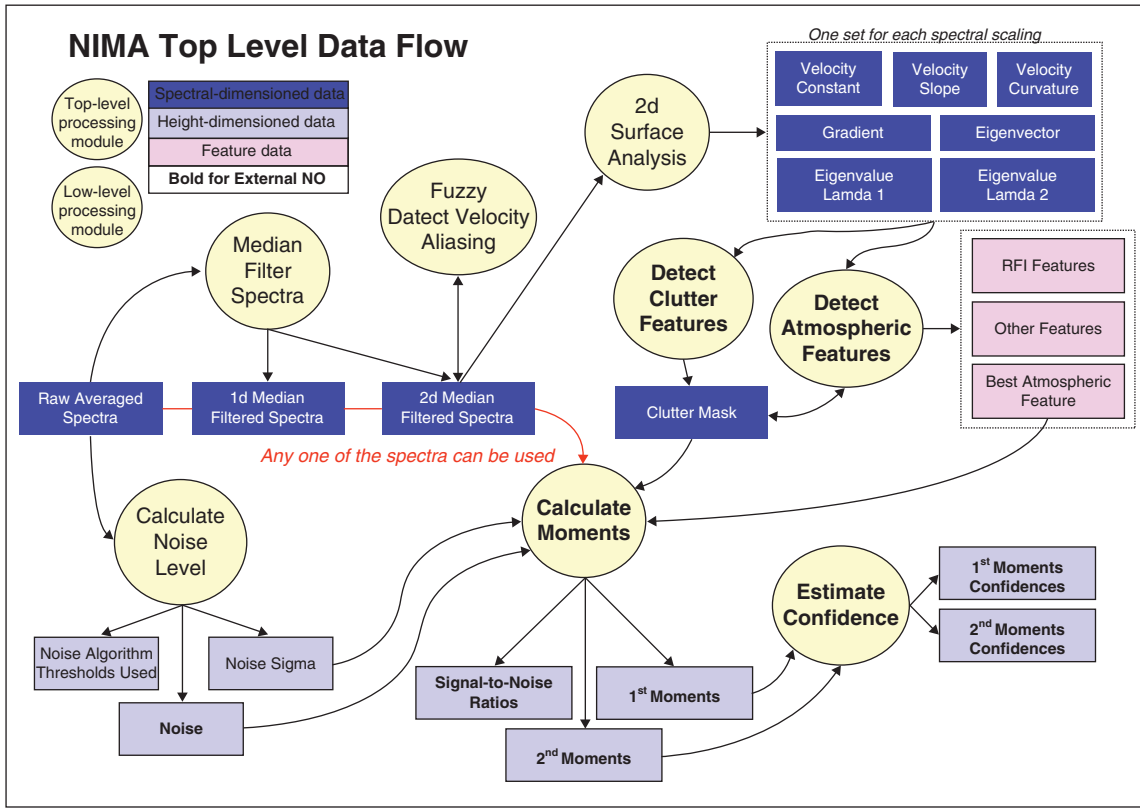


Fig. 17.2 High-level flow diagram depicting the processing elements used in NIMA. A key to interpreting the colors and lines is given in the upper left. Source: Figure provided by Cory Morse.

using that information to isolate the weather signal, reconstructing the weather signal if it overlaps one of the contamination features, and then applying (17.1) at each range gate to derive the moments. A quality control index, or "confidence", is also associated to each moment to reflect an estimate of its quality. Although we focus on the processing applied to a single beam in the simplified description here, information from other beams adjacent in space and time may also be used by NIMA to provide additional information for disambiguating feature sources.

A simplified high-level depiction of NIMA's data flow is shown in Fig. 17.2. The algorithm uses separate fuzzy inference procedures in several different steps, each focusing on a different scale or type of feature membership. Each procedure can be iterative, and the algorithm can also iterate between two steps (such as between steps four and five, below) in order to mimic a human expert's decision-making process. The five basic steps of NIMA's processing sequence are described in greater detail in the remainder of this section. Briefly, they are:

1. Prior to NIMA, the radar signal processor software averages spectra obtained over a prescribed time period to reduce noise and artifacts. NIMA performs local 2-D median filtering over a range, velocity window to eliminate outlier pixels and further improve the data, resulting in a smoothed spectral function $f(r, v)$. Both the original and smoothed spectra may be used in later steps.
2. Perform calculations (e.g., gradient, curvature, symmetry) at or around each (r, v) pixel to produce derived quantities (also called image characteristics) that are used by a fuzzy inference process to assess whether that pixel is likely associated with clutter, point targets, RFI, weather, or another phenomenon.

3. Use the results from (2) in an additional iterative fuzzy inference procedure to identify 2-D "features" (regions of interest) for the weather and contaminants.
4. Estimate the random "noise" level for each spectrum, reconstruct the weather spectrum for each range gate if it overlaps a contamination feature, and calculate its moments and associated confidences.
5. Evaluate the continuity of the moments in range, and repair egregious discontinuities using interpolation, extrapolation, and/or revisiting (4) to see if choosing a different feature as the weather signal for the suspect range gates provides a more consistent result.

The averaging and median filtering in step (1) are required to improve the quality of the spectra by reducing the magnitude of the random noise perturbations. The law of large numbers from probability theory guarantees that averaging a sufficiently large number of spectrum samples, each contaminated by (reasonably well-behaved) random noise, will converge toward the underlying true spectrum. In practice, the fact that the weather changes in time limits the number of spectrum measurements that can be meaningfully averaged, so the noise may not be adequately removed. However, the fact that a weather spectrum should usually be continuous in both r and v allows a 2-D median filter to be applied to further reduce the noise and other small-scale contaminants that may remain. The filter is applied by centering a "window", typically five velocity bins by three range bins, around each (r, v) point and replacing the value at that point, $f(r, v)$, with the median of all points in the window. Note that in a region where $f(r, v)$ is constant, linear, or even just monotonic, the median filter does not change the spectral value, but it completely removes isolated outliers or features that are small compared to the window size. The result of this operation for the data shown in Fig. 17.1 is the "cleaned up" stacked spectra shown in Fig. 17.3, which also has NIMA's final spectral moments overlaid. The large-scale features are now more coherent and somewhat more easily identified.

Step (2) begins by computing local characteristics of the stacked spectra image around each (r, v) point and then using them to determine a membership value in each pixel for "weather feature", "clutter feature", "RFI feature" or another contaminant feature. Calculations may be performed in either original spectrum units, the logarithmic decibel (dB) scale, or with renormalization of the spectrum at each range gate so that its maximum value is 1. For example, a ground clutter feature is comprised of a set of points characterized by (a) f is generally symmetric or nearly symmetric around zero velocity, (b) f has a large positive slope $\partial f(r, v)/\partial v$ to the left of zero and a large negative slope to the right, (c) f has small range slope $\partial f(r, v)/\partial r$, (d) f has a large negative velocity curvature $\partial^2 f(r, v)/\partial v^2$, (e) f has small range curvature $\partial^2 f(r, v)/\partial r^2$, (f) f has a narrow Gaussian shape, and (g) f matches well with a user-provided "clutter template". Each of these characteristics comprises "evidence" of a ground clutter feature, although none by themselves may uniquely identify it. For each pixel, each of these characteristics is mapped into a value from 0 to 1 to form a set of "interest" values or proto-membership function values that are later combined to obtain the ground clutter membership value for that pixel. For instance, the proto-membership for ground clutter based on symmetry might be written $\mu_a^{GC}(r, v)$, and the proto-membership for ground clutter based on the local match to the clutter template, $\mu_g^{GC}(r, v)$. These proto-memberships are then combined in a weighted arithmetic mean, similar to the "fuzzy consensus reasoning" described in Chapter 6, but here comprising a fuzzy logical operation similar to a softened "OR" (see Chapter 6, Section 6.3):

$$\mu^{GC}(r, v) = \frac{\alpha_a^{GC}\,\mu_a^{GC}(r, v) + \alpha_b^{GC}\,\mu_b^{GC}(r, v) + \cdots + \alpha_g^{GC}\,\mu_g^{GC}(r, v)}{\alpha_a^{GC} + \alpha_b^{GC} + \cdots + \alpha_g^{GC}} \tag{17.2}$$

for various coefficients α between 0 and 1. Note that if one or more of the characteristics (a)–(g) is not available at a given pixel, the corresponding coefficients α may be set to 0 and (17.2) will still give the consensus membership based on the remaining characteristics. Both the proto-membership functions and the coefficients are tuned based on the profiler location and the characteristics of clutter there. For

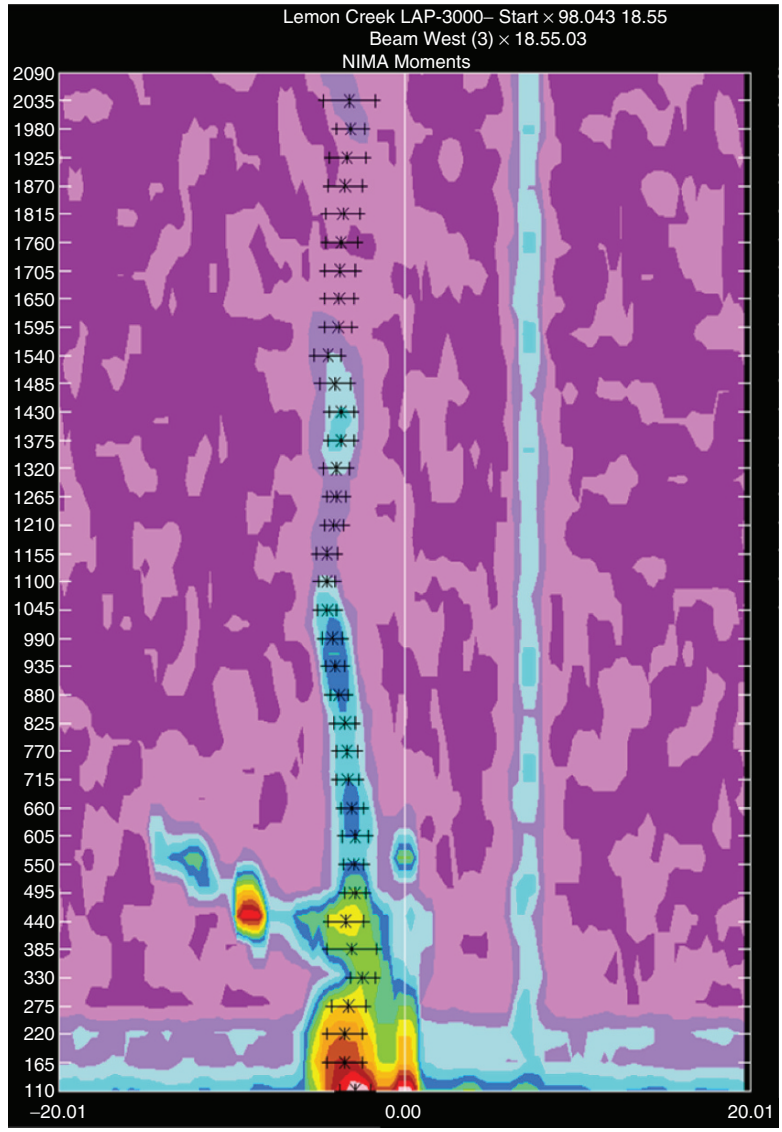


Fig. 17.3 Stacked spectra plot for the same profiler beam shown in Fig. 17.1. The spectra have been median-filtered, and the overlaid moments are from NIMA. Source: Adapted from Fig. 1 of Morse et al. (2002).

instance, the developers of NIMA report that typical sea clutter contamination tends to be shifted from 0 ms^{-1} in velocity by an amount that varies with the site. Membership functions for point targets, RFI, weather, and other phenomena are computed similarly and also tuned for each site.

In step (3), 2-D sets of points, or "features", are identified and refined based on the pointwise membership values for weather and clutter, point targets, RFI, and other contaminants. The process starts by constructing a "proto-feature" for each phenomenon based on the set of all points having membership values above a certain threshold. For example, the RFI proto-feature might be estimated as $\left\{(r, v)|\mu^{RFI}(r, v) > T_{RFI}\right\}$, where T_{RFI} is a membership threshold value between 0 and 1, say 0.5. Each proto-feature has various characteristics that may be used to determine whether or not it is really associated with RFI. A true RFI feature should have (a) a medium-sized average width in the velocity coordinate, (b) low variance of the width over the range gates in the feature, (c) small variance of the width when normalized (i.e., divided by) the number of range gates in the feature, (d) low variance in the

peak velocity over the range gates in the feature, (e) low variance in the velocity midpoint over the range gates in the feature, (f) small variance of the velocity midpoint when normalized by the number of range gates in the feature, (g) a small absolute slope of the velocity midpoints with range and (h) a location close to a (temporally trended) RFI feature identified in the previous measurement. As with the original pixel identification, each of these characteristics may be used to compute proto-membership values $\mu_a^{RFI\,feature}(n)$, …, $\mu_h^{RFI\,feature}(n)$, where n represents the proto-feature number. These values may be combined via a weighted sum as in (17.2). However, it is also known that a feature is likely to be from RFI only if (i) the number of range gates covered by the feature is large and (j) the average difference over range gates between the midpoint velocity and the RFI velocity previously measured is small. If either of these characteristics is not present, the RFI membership value of the proto-feature must be suppressed. For this purpose an additional proto-membership function $\mu_i^{RFI\,feature}(r, v)$ is defined in such a way that it is 0 for "small" numbers of range gates in the proto-feature and 1 for "large" numbers, and $\mu_j^{RFI\,feature}(r, v)$ is defined to be 1 for velocities v "far" from the velocity measured in the previous timestep, and to have a value near 0 otherwise. These two proto-membership function values are combined together with the consensus from the other proto-membership functions using a geometric average, which was introduced as a soft version of "AND" in Chapter 6, Section 6.3 (see Fig. 6.6). The membership of the n^{th} proto-feature in the set of RFI features is then given by the following formula, in which the "RFI feature" superscripts have been omitted on the right-hand side for conciseness:

$$\mu^{RFI\,feature}(n) = \left(\frac{\alpha_a\,\mu_a(n) + \;\cdots\; + \alpha_h\,\mu_h(n)}{\alpha_a + \;\cdots\; + \alpha_h}\right)^{\alpha_A} \cdot \left(\mu_i(n)^{\alpha_i} \cdot \mu_j(n)^{\alpha_j}\right)^{\alpha_G} \tag{17.3}$$

for some coefficients $\alpha_a, \ldots, \alpha_j$, α_A and α_G; NIMA uses $\alpha_A = (\alpha_a + \alpha_b + \cdots + \alpha_h)\big/(\alpha_a + \alpha_b + \ldots + \alpha_j)$ and $\alpha_G = 1\big/\left(\alpha_a + \alpha_b + \cdots + \alpha_j\right)$ so that the sum of all the exponents is 1. Note that if either $\mu_i(n) = 0$ or $\mu_j(n) = 0$ is zero, then $\mu^{RFI\,feature}(n) = 0$; otherwise, their impact is modulated by the choice of exponents. A similar approach generates membership values for each proto-feature for the various different possible sources: clutter, point targets, weather, etc. After the proto-features are generated, they are evaluated by yet another fuzzy logic procedure to judge whether they are adequately coherent. For instance, if a proto-feature for "weather" contains a number of "holes" or is suspiciously narrow, the threshold $T_{weather}$ might be lowered and step (3) repeated. In addition, disjoint weather proto-features may be joined so that the weather feature covers all range gates. These processes may be iterated several times until a satisfactory set of features is identified.

In step (4), the "best" weather signal is identified and used as the basis for computing the Doppler moments. For each range gate, the spectral values at the velocities falling within the weather signal feature may be used to compute the moments via (17.1), so long as there is no overlap with other features. If there is overlap, NIMA uses another fuzzy-logic procedure to identify the region around the peak spectral value within the weather signal that may safely be associated predominantly with the weather, and uses a Gaussian fit to those points to replace the suspect spectral values. For each range gate, the random "noise" level and its variance are calculated using the method of Hildebrand and Sekhon (1974). This technique also generates statistics, including the "signal to noise" ratio, that are used along with the quality of a Gaussian fit to the weather signal, the difference between the Gaussian moments and direct moment calculations, and the local continuity of the resulting moments in range in constructing a fuzzy membership function providing the "confidence" in each moment for each range gate. This confidence is a number between 0 and 1 that assesses the quality of the estimate, though not in a precise statistical sense.

Finally, step (5) evaluates the moment confidences for each range gate to identify intervals of low-confidence, suspect values that should be "repaired". For range intervals in which this confidence is small, NIMA repeats step (4) using a "second guess" at the location of the weather signal. The location may be chosen using the "second best" weather feature at that range or by using the moments from neighboring "good" ranges to interpolate a "first guess" weather

signal velocity to use in place of the peak spectral value for reconstructing the weather signal using a Gaussian fit. This process may be repeated more than once until a "best" set of moment estimates are identified.

17.2.4 Summary

The NIMA algorithm provides a good example of how a fuzzy logic procedure can be constructed to duplicate a human expert approach to a complicated problem. It applies both local and global image analysis, uses iteration to improve initial results, and provides a "confidence" value in each of the final moment estimates. Such confidences are very valuable to downstream applications. For instance, the original motivation for developing NIMA was to provide improved rapid-update wind-profiler measurements of horizontal winds for use in detecting wind shear and terrain-induced turbulence hazardous to aviation along the approaches to the Juneau, Alaska airport (Goodrich et al. 2002). The wind profiler points nearly vertically, measuring radial velocity as it tilts slightly to the north, south, east and west. These radial velocities are then used to calculate a best-fit horizontal wind at each level above the ground. By providing confidences associated with each velocity measurement, the horizontal wind computation is able to weight each velocity measurement appropriately; only three good radial velocity measurements at each level are necessary to estimate the horizontal wind. Moreover, the confidences are used by the horizontal wind algorithm to produce a confidence value for the final wind measurement. Not only does this use of confidence values improve the overall quality of the wind measurements, but the confidences also propagate into the hazard warning system so that the likelihood that contamination could cause a false aviation hazard alert is greatly reduced. The practical benefit is that unnecessary airspace closures, rerouting, and delays may be significantly diminished.

17.3 Operational Radar Data Quality Control

Ground clutter contamination within weather radar moment data has two sources: normal propagation (NP) ground clutter return from stationary targets such as buildings, power lines, or terrain, and anomalous propagation (AP) ground clutter return that arises from particular atmospheric conditions within the planetary boundary layer (i.e., temperature inversions and strong humidity gradients) that bend or duct the radar beam towards the ground. In the United States, conditions conducive to AP are most prevalent during summer months (Steiner and Smith 2002). Because NP targets are stationary and unchanging for long periods of time, they are relatively easy to remove through the use of clutter maps that direct where clutter filters are applied. However, AP clutter return can evolve and dissipate as atmospheric conditions change, thus necessitating automatic detection and removal since a fixed clutter map does not include AP clutter locations.

Radar base data contamination from the sea surface occurs when the radar beam grazes the sea surface or when strong return is captured through the beam's sidelobes. The sea state mostly determines the amount of sea clutter detected and is influenced by factors such as wave height, wind speed, wind direction relative to the direction of the radar beam, the direction of the waves relative to the direction of the radar beam, whether the tide is incoming or outgoing, or the presence of swells or waves, to name a few (Skolnick 1980). Radar system characteristics can also have an effect on the amount of sea clutter detected. Further, under AP conditions, the range over which sea clutter occurs can be considerably extended.

While an adaptation of the NIMA method described in the previous section might in principle be used to identify and automatically remove ground and sea clutter during the radar signal processing, such a system has not yet been implemented on operational Doppler weather radars in the U.S. Therefore, clutter contaminant removal within the radar base moment data (reflectivity, radial velocity, and spectrum width) is necessary to ensure that downstream analysis algorithms are provided only high quality output. Using a consensus fuzzy logic methodology (see Chapter 6, Section 6.8), the Radar Echo Classifier (REC; Kessinger et al. 2003) identifies various categories of scatterer at each radar range gate to facilitate the removal of NP and AP ground clutter as well as sea clutter contamination. The REC has a modular design and consists of three algorithms: the Anomalous-Propagation ground clutter Detection Algorithm (REC-APDA), the Precipitation Detection

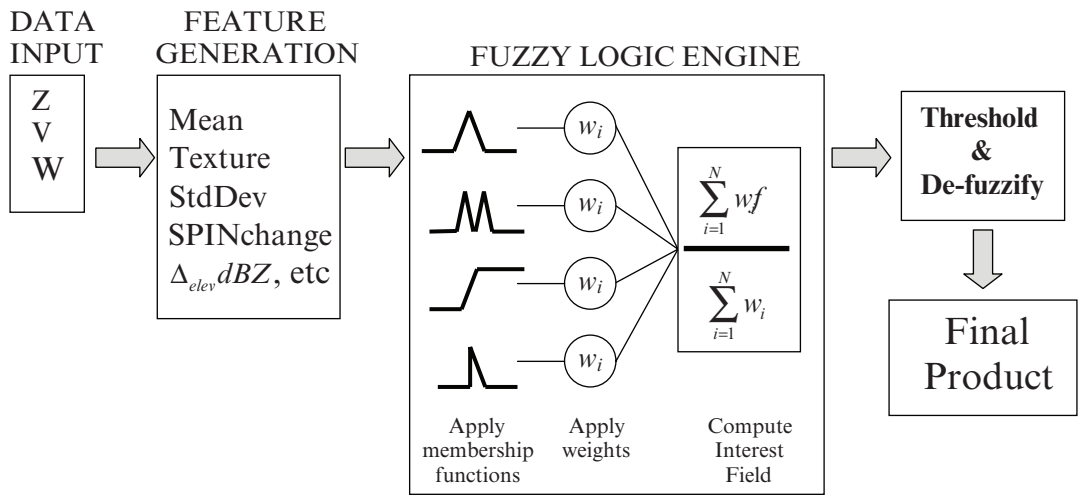


Fig. 17.4 Schematic showing the processing steps within the Radar Echo Classifier. These processing steps include: ingesting the base data for reflectivity (Z), radial velocity (V), and spectrum width (W); generation of features that are derived from the base data fields; use of a fuzzy logic "engine" to determine the initial interest output; application of the appropriate threshold and de-fuzzification; and the final output product for the type of radar echo being considered.

Algorithm (REC-PDA) and the Sea Clutter Detection Algorithm (REC-SCDA). Initial development of the REC-APDA was accomplished by Cornelius et al. (1995) and Pratte et al. (1997) with additional refinements added by Kessinger et al. (2003). Radar base moment fields of reflectivity, radial velocity and spectrum width are inputs for each REC algorithm.

The REC-APDA was deployed within the NWS WSR-88D (Crum and Alberty 1993) Radar Product Generator (RPG; Saffle and Johnson 1997; Crum et al. 1998; Reed and Cate 2001) system in September 2002 as part of the AP Clutter Mitigation Scheme (Keeler et al. 1999). The REC-APDA removes AP ground clutter contamination from the reflectivity field before the calculation of the radar-derived rainfall rate. The REC-SCDA and a modified REC-APDA were deployed on the United Arab Emirates (UAE) radar network in October 2002 (NCAR Research Applications Program 2003).

17.3.1 Schematic of REC Algorithms

Each REC module processes the input fields of radar reflectivity (dBZ), radial velocity (m s^{-1}) and spectrum half-width (m s^{-1}) to identify the type of scatterer that is present within each range gate, whether it is a "ground clutter target", a "precipitation target" or a "sea clutter target" (Fig. 17.4). As in NIMA, various characteristic (or "feature") fields are then computed and mapped into a value between zero and unity to obtain the "interest" or proto-membership function values that are combined in a weighted, normalized arithmetic mean (similar to "fuzzy consensus reasoning" described in Chapter 6) to ascertain the likelihood that the scatterer of interest is present. After application of the appropriate threshold value, the final binary output is the detection product that indicates whether or not the scatterer type of interest is present at a particular range gate.

17.3.2 Methodology for Creating a REC Algorithm

In this section, the step-by-step procedures for devising a detection module for the REC are described. Following the REC schematic described above, the first step is to calculate all potential characteristic (or feature) fields that might be used to detect the radar scatterer of interest. Next, using histograms, each feature field is quantified through comparisons with validation data sets to ascertain the best discriminators of the desired scatterer type from other types. After the feature fields providing the best discrimination are selected, the histograms are used to devise the membership functions to scale the feature fields between zero and unity, in accordance with their

distributions for the different scatterer types. Each step is described below.

17.3.2.1 Feature Field Calculation

First, the radar moment fields (reflectivity, radial velocity and spectrum width) are input into the "feature generator" (Fig. 17.4) to calculate derived quantities such as the "texture", mean, standard deviation, median, and the "*SPINchange*" (Steiner and Smith 2002). These derived quantities can be applied to any or all of the radar's moment fields. For the WSR-88D, they are calculated over a local area that typically encompasses a data kernel of 3 adjacent beams and 8 or 16 range bins. For a given moment X, the texture, mean and standard deviation are,

$$Texture(X), T_X = \left(\sum_{j=1}^{Nbeams} \sum_{i-1}^{Ngates} \left(X_{i,j} - X_{i-1,j} \right)^2 \right) \Big/ (Ngates \times Nbeams) \tag{17.4}$$

$$Mean(X), \bar{X} = \left(\sum_{j=1}^{Nbeams} \sum_{i=1}^{Ngates} X_{i,j} \right) \Big/ (Ngates \times Nbeams) \tag{17.5}$$

$$Standard\ Deviation(X), \sigma_X = \left[\left(\sum_{j=1}^{Nbeams} \sum_{i=1}^{Ngates} (X_{i,j} - \bar{X}) \right)^2 \Big/ (Ngates \times Nbeams - 1) \right]^{0.5} \tag{17.6}$$

where Ngates is the number of range gates and Nbeams is the number of adjacent beams within the local area.

The *SPINchange* variable indicates the percentage within the data kernel of changes in sign of the gate-to-gate reflectivity difference field, $dBZ_{i,j} - dBZ_{i-1,j}$, that exceed a minimum difference threshold, dBZ_{thresh}. While Steiner and Smith used a value of $dBZ_{\text{thresh}} = 2.0$, the value of dBZ_{thresh} was set to 3 dBZ within the REC-APDA due to differences in the implementation. They also formatted the reflectivity data within Cartesian space whereas the REC-APDA retains the reflectivity data in polar coordinates.

Within sea clutter radar return, the vertical gradient of reflectivity is typically large with a rapid decrease in value as vertical distance from the ocean's surface increases. The gate-to-gate difference in reflectivity values between adjacent elevation angles ($\Delta_{elev}dBZ$) is used to aid in the discrimination of sea clutter return and is calculated as follows:

$$\Delta_{elev}dBZ = dBZ_u - dBZ_l \tag{17.7}$$

where dBZ_u is the reflectivity value at the upper elevation angle, dBZ_l is at the lower elevation angle and Δ_{elev} indicates their difference.

The $\Delta_{elev}dBZ$ variable is used three ways. First, it is used directly to discriminate sea clutter. Second, stratiform precipitation at long radar ranges will also have large values of $\Delta_{elev}dBZ$ due to the shallow extent of the precipitation echo. To account for stratiform situations, the $\Delta_{elev}dBZ$ field is multiplied by the range weighting function (RangeWgt; shown in Fig. 17.5) as specified in (17.8) to create the range-weighted vertical difference in reflectivity ($R\Delta_{elev}dBZ$) field.

$$R\Delta_{elev}dBZ_i = \Delta_{elev}dBZ_i \times RangeWgt_i \tag{17.8}$$

where i is the range gate number. The scaling factor $RangeWgt_i$ is chosen to reduce large vertical reflectivity differences at long ranges.

Third, $\Delta_{elev}dBZ$ is divided by the sine of the angular difference between the two elevation angles to calculate the vertical gradient of the reflectivity field,

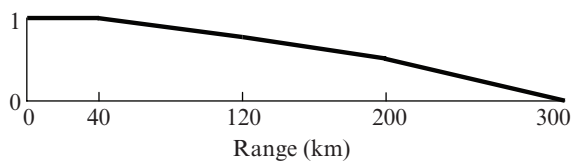


Fig. 17.5 Range weight function (RangeWgt) used in calculating the feature field $R\Delta_{elev}dBZ$.

$\frac{\Delta dBZ}{\Delta z}$, as below

$$\left(\frac{\Delta dBZ}{\Delta z}\right)_i = \Delta_{elev} dBZ_i \Big/ [\sin(\theta_u - \theta_l)] \qquad (17.9)$$

where Δz is the change in vertical distance, θ_u is the upper elevation angle, θ_l is the lower elevation angle, and i is the range gate number along the beam.

17.3.2.2 Data Characterization Through the Use of Validation Data Sets

Choosing those feature fields that best discriminate between the scatterer type of interest and all others is a critical first step in designing a fuzzy logic consensus scheme. Understanding the nature of the feature fields allows for the selection of those fields with best performance at discrimination. Using a validation data set (or validation field) informs the selection process. Two methods for devising a validation field are used here: a human expert manually defines all scatterer types present in the radar scan, or an independent identification algorithm is used. Use of an independent algorithm can be preferable to manual definition because it can be more easily accomplished and allows many more radar scans to be examined; however, this algorithm must have excellent performance at discriminating scatterer types. The manual definition process used by an expert requires the use of a radar data-editing program such as the NCAR SOLO program (Oye et al. 1995) and can be a time-consuming and tedious process.

For manual validation, the expert defines the validation categories by examining the various radar fields and their derived products. All regions of the lowest elevation scan (typically 0.5-degree) are characterized as ground clutter, precipitation, sea clutter, clear air return (typically from insects), or "other". Figure 17.6 shows an example of an expert validation field that was manually drawn. Typically, ground clutter (whether from NP or AP sources) is characterized by a near-zero radial velocity, low spectrum width, and high texture of the reflectivity field, among others. The expert can examine the scan at higher elevation angles to determine if the radar echo in question has vertical continuity. A lack of vertical continuity does not guarantee that the echo is AP clutter, but gives more information into the decision-making process. Once the expert has determined where the AP clutter is, a polygon is drawn to set the appropriate value of the validation field.

A limitation of defining the validation categories in this manner is the lack of independence between the validation field and the radar fields used as input into the REC, because the fields used by both are the same.

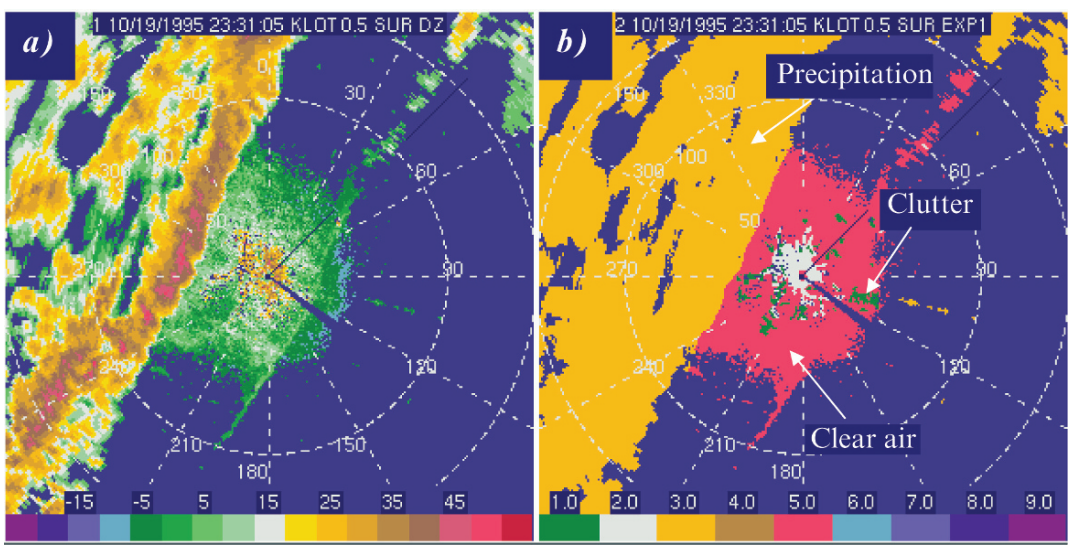


Fig. 17.6 An example of the subjective validation data set as drawn by human experts on WSR-88D data from Chicago, IL (KLOT). Fields shown include (a) the reflectivity (dBZ) and (b) the expert validation field. Validation categories are shaded such that gold is precipitation, red is clear air return from insects, green is ground clutter, and white is unclassified return. The 0.5-degree elevation angle is shown with range rings at 50 km intervals.

Table 17.1 The radar scatterers identified by the Particle Identification (PID) algorithm are listed in the left column, with the assigned PID category number in the center, and with the corresponding Radar Echo Classifier (REC) algorithm that uses the PID output as the objective validation field in the right column.

PID category	PID number	Used as validation for
Light rain	3	REC-precipitation detection algorithm
Moderate rain	4	
Heavy rain	5	
Hail	6	
Rain and hail mixture	7	
Graupel and small hail	8	
Graupel and rain	9	
Dry snow	10	
Wet snow	11	
Ice crystals	12	
Irregular ice crystals	13	
Super-cooled liquid droplets	14	
Ground clutter	17	REC-AP detection algorithm

A preferable methodology is to have an independent data source for determination of truth regions. The data available from the NCAR S-Pol dual-polarimetric radar (Keeler et al. 2000) provides an opportunity to obtain an independent determination of the validation field through use of the Particle Identification (PID) algorithm (Vivekanandan et al. 1999). The additional measured fields of dual-polarimetric radars not only provide the desired independent data, but also make robust ground clutter determination relatively easy. At each range gate, the PID uses polarimetric and moment data to classify the hydrometeor type using a fuzzy logic methodology that also includes categories for ground clutter and insects. An input sounding determines the freezing level to separate liquid from frozen hydrometeor types. Echo types that are classified by the PID and used as the validation field for a particular REC algorithm are listed in Table 17.1.

To construct the validation field for evaluation of the REC algorithms, PID categories can be grouped. For the APDA, the one category of "ground clutter" defines the validation field. For the PDA, the categories from "light rain" through "irregular ice crystals" are combined to define the location of precipitation echoes. Sea clutter is not yet a category within the PID. Comparison of Fig. 17.7a (shows all the PID categories) to Fig. 17.7b (precipitation categories have been combined) illustrates how the precipitation PID categories are combined to form a validation region (i.e., gold region in Fig. 17.7b).

17.3.2.3 Histograms of Data Characteristics

Histogram plots of the feature fields are made using the validation field to distinguish between the types

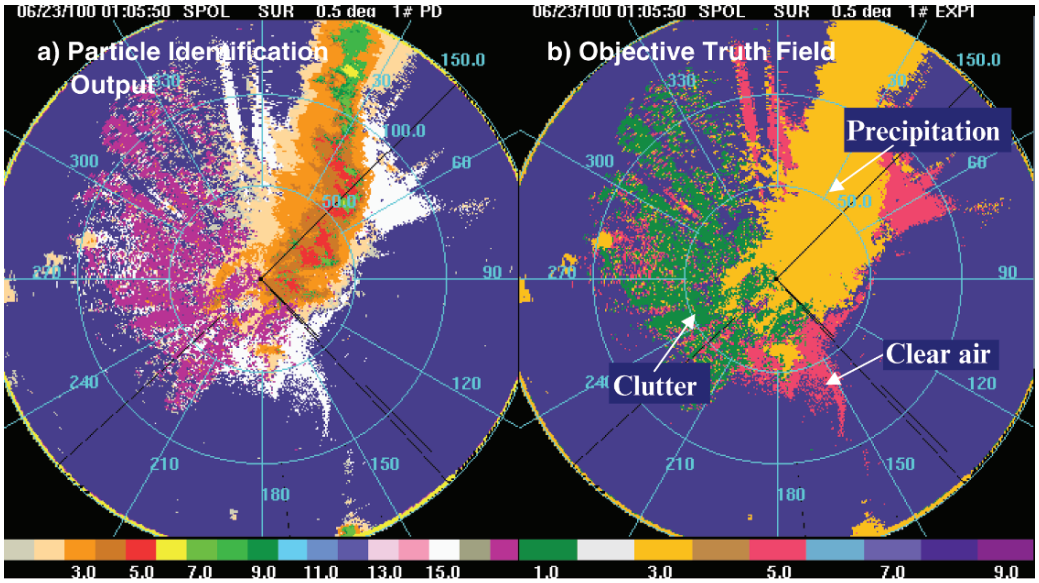


Fig. 17.7 Conversion of the multiple categories of (a) the Particle Identification (PID) algorithm output is accomplished to create (b) an objective validation field for NCAR S-Pol data. Table 17.1 lists the category numbers for each color value in (a). See Fig. 17.6 for the color key for (b). Data on the 0.5-degree elevation scan are shown.

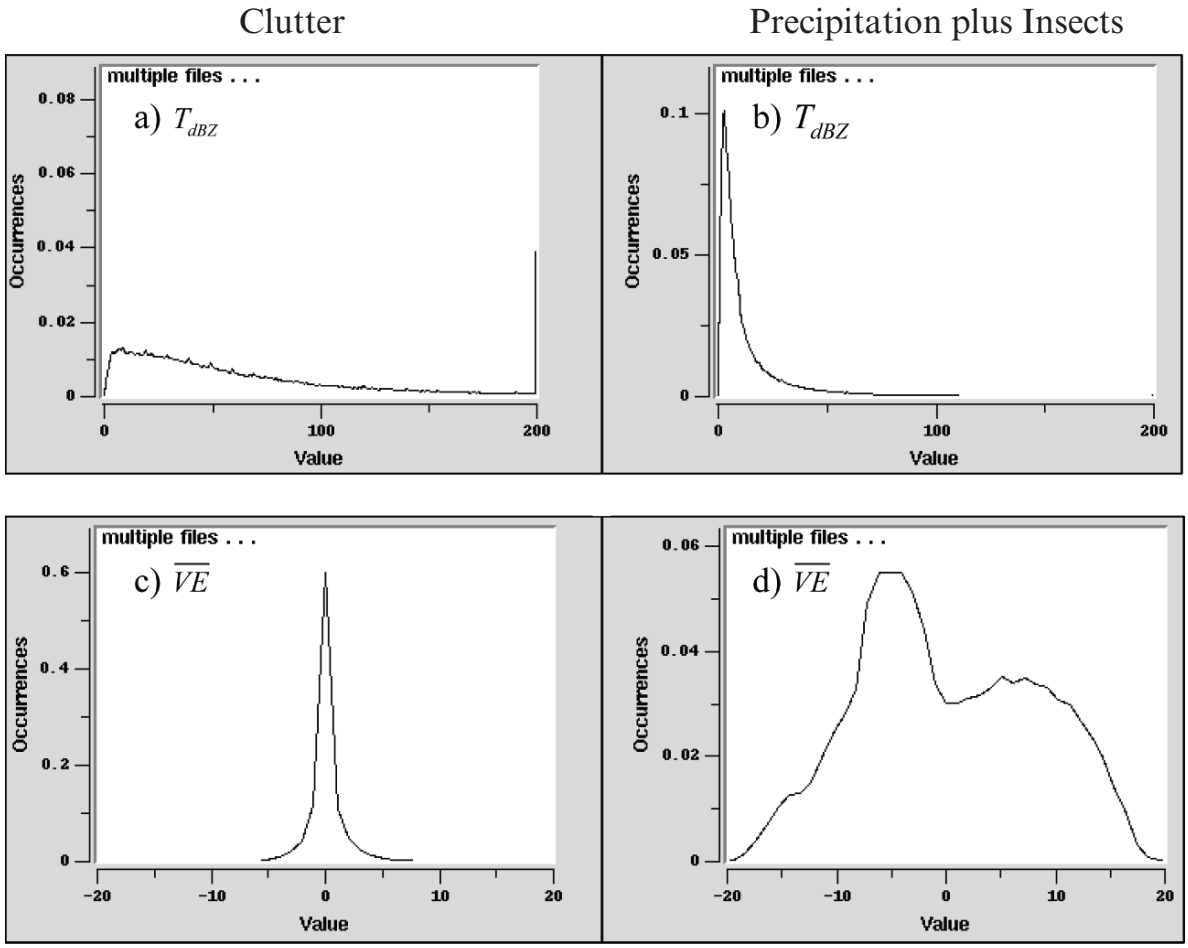


Fig. 17.8 Histogram plots showing the frequency of range gates containing each feature field within clutter (left column) and within precipitation plus clear air return from insects (right column) as determined by the validation field. Fields shown are (a and b) texture of the reflectivity (T_{dBZ}; dBZ^2) and (c and d) the mean radial velocity field ($\overline{VE}$; m s^{-1}). Source: Data were taken from the Dodge City, KS, WSR-88D (KDDC) at 0.5 degree elevation angle.

of radar echoes. Figure 17.8 shows examples of histograms of two variables used in the REC-APDA as derived from data taken by the Dodge City, KS (KDDC) WSR-88D radar. Fields shown include the texture of the reflectivity (T_{dBZ}) and the mean radial velocity ($\overline{VE}$). These figures show that ground clutter is characterized by high values of T_{dBZ} and by near-zero values of $\overline{VE}$ while precipitation plus return from insects is characterized by low values of T_{dBZ} and by a broad distribution of $\overline{VE}$ values. These two feature fields are good discriminators of ground clutter from other echo types.

Histograms are computed in this manner for each potential feature field and used to find the best discriminators for the echo type of interest. Feature fields with the best discrimination performance are used in the REC modular algorithms.

17.3.2.4 Membership Functions

After the appropriate feature fields are selected, the histograms are used to devise the membership functions that scale the values between zero and unity and output the "interest" field. Within the interest field, a zero value indicates "no likelihood" that the feature field is associated with the radar return type of interest, while unity indicates a "high likelihood".

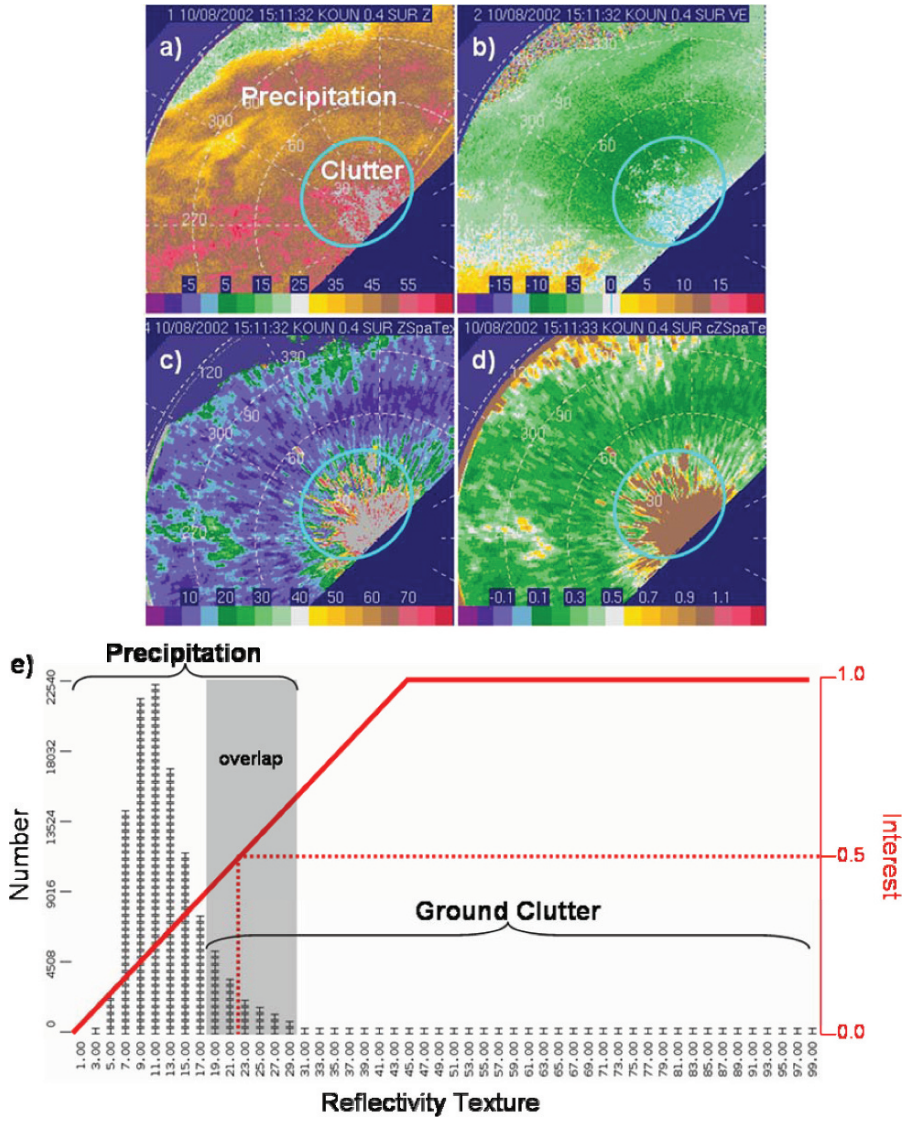


Fig. 17.9 Use of a histogram to design a membership function to detect ground clutter. The radar base moment fields of (a) reflectivity (dBZ) and (b) radial velocity (m s^{-1}) are shown with regions of precipitation and ground clutter (blue oval) indicated. Radial velocity values near zero are shaded cyan. A feature field, the (c) reflectivity texture field (T_{dBZ}; dBZ^2), is shown where grey values within the oval are >80 dBZ^2. From the data in (c), a histogram (e) is computed that shows the distribution of values. A stepwise linear membership function is drawn (red line) from the histogram and applied to (c) to scale the values between zero and unity, with the resultant interest field shown in (d).

Figure 17.9 illustrates how the values within the histogram can be used to devise the membership function.

From the base moment data of reflectivity (Fig. 17.9a), the feature field of reflectivity texture (T_{dBZ}) is calculated using (17.4) with the result shown in Fig. 17.9c. The corresponding radial velocity field is shown (Fig. 17.9b) for reference purposes only. From the reflectivity texture field, a conditional histogram is computed that shows the distribution of values for different echo sources. Regions of low texture are generally precipitation while regions of high texture are generally ground clutter. Notice that there is a region of overlap (indicated by the grey shading) where the reflectivity texture values could indicate either precipitation or ground clutter.

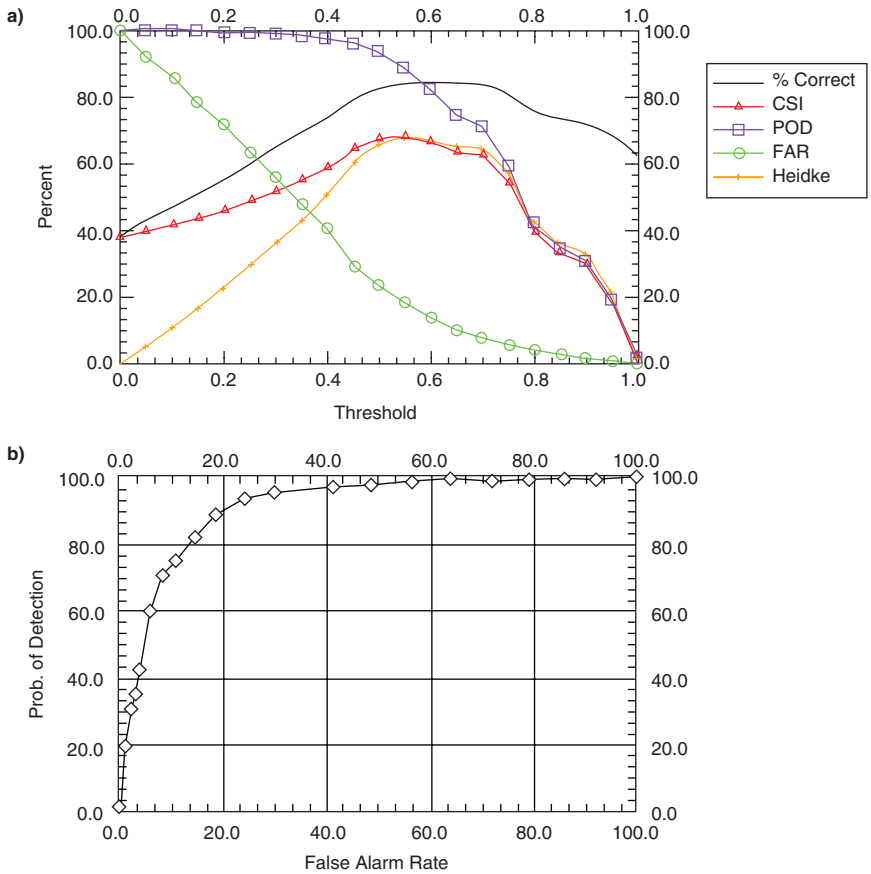


Fig. 17.10 Statistical performance curves for the REC-APDA as computed using only one elevation scan from the NCAR S-Pol radar. In (a), performance curves of percent correct (%Correct), Critical Success Index (CSI), Probability of Detection (POD), False Alarm Ratio (FAR) and the Heidke Skill Score (Heidke) are shown at increasing values for the de-fuzzification threshold and indicates that optimal performance is achieved at the 0.55 threshold. In (b), the ROC curve is shown for the same scan.

Using the validation field, the statistical skill of the interest field realized after application of the membership function can be determined. Statistical skill scores such as the percent correct, the Critical Success Index (CSI), the Heidke Skill Score (HSS), the probability of detection (POD), the false alarm ratio (FAR) (Donaldson et al. 1975, Wilks 1995; see also Chapter 2 of this volume) among others can all be used to measure performance. Interest field performance is examined by increasing the applied threshold (such as $0, 0.1, 0.2 \ldots 1.0$) to reveal where discrimination performance is maximized for any skill score used for optimization (Fig. 17.10a) and allows a Relative (or Receiver) Operating Characteristic (ROC, Hanley and McNeil 1982; see also Chapter 2) plot to be drawn (Fig. 17.10b). REC-APDA output is shown (Fig. 17.10) from one scan of S-Pol radar data where the PID algorithm is used as the validation data set. Maximum skill scores are obtained by the CSI and the HSS at a threshold value of 0.55, indicating the preferred threshold value for this one scan. The optimal performance threshold value must be determined with many more scans. Using these types of plots, the membership function for each feature field can be adjusted until the statistical performance is maximized near an interest value of 0.5, the midpoint interest value. This ensures that each feature field has optimum performance near the same threshold value.

17.3.2.5 Weight Selection and De-fuzzification

Each interest field is assigned a weight that reflects its level of skill to detect the scatterer of interest. Using the validation field, the statistical performance of each interest field is used to devise the appropriate weight, in a similar methodology as for tuning the membership functions. This "de-fuzzification" step reduces the interest field to a binary indicator for the presence/absence of the desired scatterer type. To accomplish this, an appropriate threshold is selected and applied to the mean weighted sum of the input

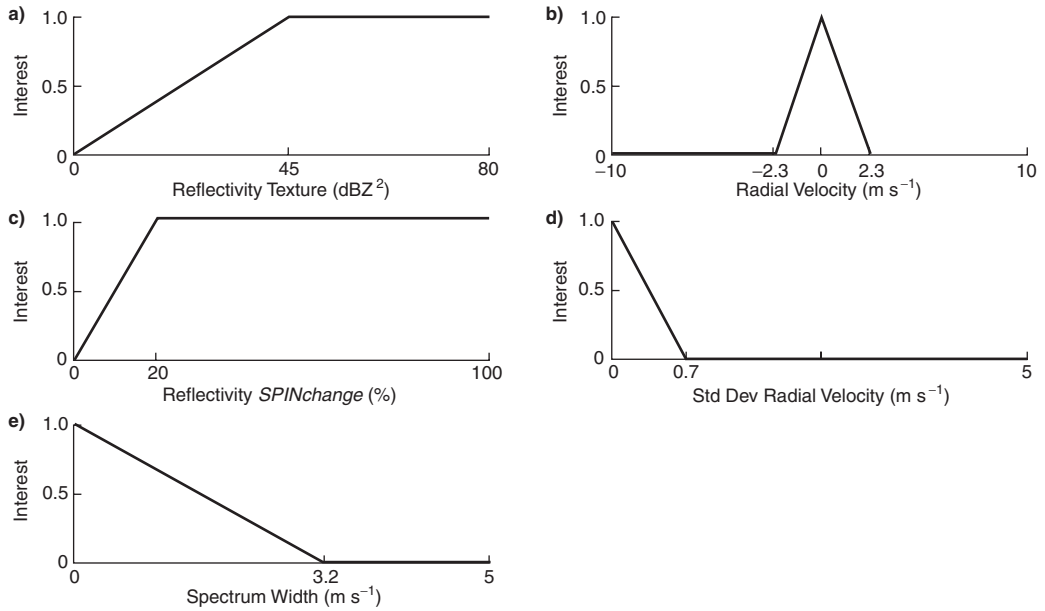


Fig. 17.11 Membership functions for each of the feature fields within the Radar Echo Classifier AP ground clutter Detection Algorithm (REC-APDA). Membership functions shown are for (a) reflectivity texture (T_{dBZ}; dBZ^2), (b) radial velocity (m s^{-1}), (c) reflectivity *SPINchange* (%), (d) standard deviation of the radial velocity (m s^{-1}), and (e) spectrum width (m s^{-1}).

interest fields. The threshold is typically selected as the midpoint (i.e., 0.5 interest), the same value for which the individual membership functions are tuned. Therefore if the mean weighted sum of interest values is greater (less) than 0.5, the binary output = 1 (0) indicating the radar return type in question is detected (not detected).

17.3.3 REC AP Ground Clutter Detection Algorithm (REC-APDA)

Under NP conditions, ground clutter from stationary targets is removed by application of ground clutter filters at pre-determined locations on a clutter map. Ground clutter filters are not universally applied because they negatively bias reflectivity values in precipitation whenever the radial velocity is near zero. Stratiform rain or snow returns are especially biased in this situation. Under AP conditions, ground clutter contamination can evolve rapidly, preventing the formulation of a clutter map. Until recently, the WSR-88D system did not have an automated method for identification of AP ground clutter. When AP conditions are present, radar operators must apply clutter filters to the contaminated areas by manual selection. To facilitate the automated identification and removal of ground clutter during AP conditions, the REC-APDA was implemented within the WSR-88D system.

The REC-APDA uses the following feature fields: the texture of the reflectivity, the radial velocity, the reflectivity *SPINchange* using a reflectivity threshold of 3 dBZ, the standard deviation of the radial velocity and the spectrum width. As implemented within the WSR-88D system, the membership functions for each feature field are shown in Fig. 17.11. A weight of unity is applied to each interest field in the computation of the weighted mean sum.

17.3.4 REC Precipitation Detection Algorithm (REC-PDA)

Because few, if any, algorithms achieve 100% detection efficiency, a Precipitation Detection Algorithm (REC-PDA) was devised to improve the discrimination between precipitation and ground clutter.

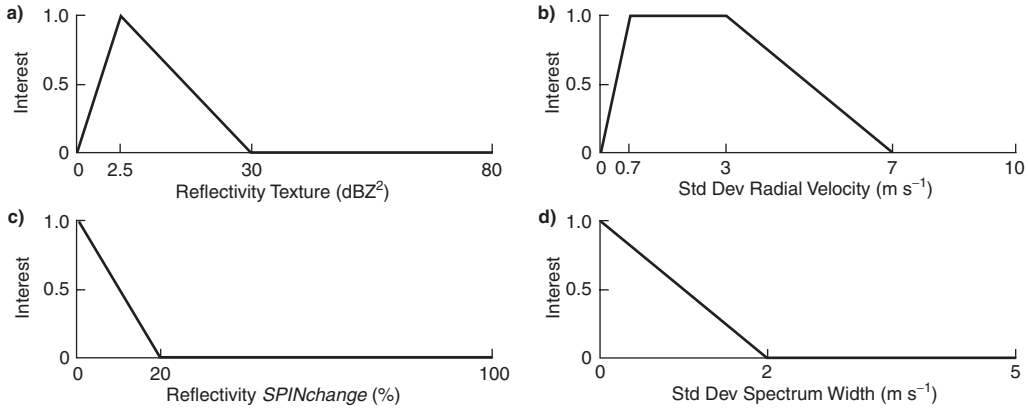


Fig. 17.12 Membership functions for each of the feature fields within the REC-PDA. Membership functions shown are for (a) reflectivity texture (T_{dBZ}; dBZ^2), (b) standard deviation of the radial velocity (m s^{-1}), (c) reflectivity *SPINchange* (%) and (d) standard deviation of the spectrum width (m s^{-1}).

The REC-PDA is designed to detect both stratiform and convective precipitation return without distinction.

The REC-PDA uses the following feature fields derived from a local area around each range gate: the texture of the reflectivity, the standard deviation of the radial velocity, the reflectivity *SPINchange* using a reflectivity threshold of 3 dBZ and the standard deviation of the spectrum width. The membership functions for each feature field are shown in Fig. 17.12. In the computation of the weighted mean sum, a weight of unity is applied to the first three interest fields and a weight of 0.25 is applied to the standard deviation of the spectrum width interest field. For summertime convective storm situations, a final threshold is applied to remove all range bins where the reflectivity is $\leq$18 dBZ. For wintertime, stratiform situations, the reflectivity threshold is set to $\leq$ 0 dBZ.

17.3.5 Application of the REC-APDA and the REC-PDA

The REC-APDA and the REC-PDA can be used in conjunction to enhance the discrimination between AP ground clutter contamination and precipitation such that data quality is improved. A case from the Boston, MA WSR-88D (KBOX) is used to illustrate the improvement. Figure 17.13 shows the reflectivity and velocity fields where the precipitation echo is to the south and the AP and NP ground clutter return is to the north of the radar. Output from the REC-APDA and the REC-PDA (Fig. 17.14) shows that both algorithms perform well at detecting their respective radar return types. The REC-APDA detects "pure" clutter (REC-APDA $>$ 0.5) while the REC-PDA detects "pure" precipitation return (REC-PDA $>$ 0.5). Base data are determined to be clutter or clutter-biased (precipitation mixed with clutter) if one of the two conditions is met: the REC-APDA $>$ 0.5 or the REC-PDA $<$ 0.5. Using both rules results in the removal of clutter regions and regions of clutter mixed with precipitation from the Hybrid Scan Reflectivity (HSR), a NEXRAD product. Figure 17.15 shows the HSR as computed with (a) only the REC-APDA used to threshold the data and (b) with the REC-APDA and REC-PDA used in combination. A clear improvement in performance is realized with the second method as the clutter return to the north of the radar is greatly reduced. This means that radar rainfall estimates will be of higher quality. A second example (Fig. 17.16; note the difference in color tables in the two rainfall panels) shows the HSR and the 4-h accumulated rainfall that result from using only the REC-APDA and the REC-APDA/REC-PDA in combination. For the first case, the maximum erroneous rainfall was equal to 3.68 in. while erroneous rainfall estimates were much reduced for the second case. The improvement stems from the REC-PDA identification of clutter mixed with clear air echoes as not being precipitation and their subsequent removal. The REC-APDA correctly identified those echoes as not being pure ground clutter.

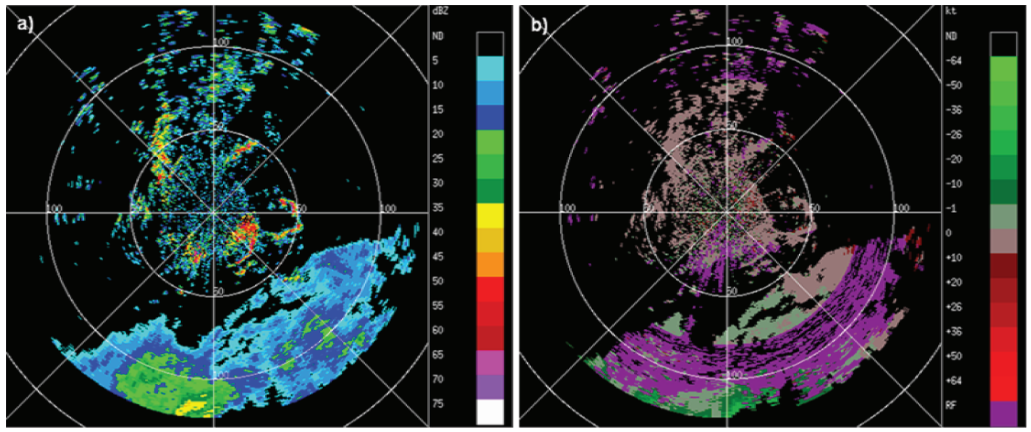


Fig. 17.13 Radar base moment fields from the Boston WSR-88D (KBOX) on 14 July 2003 at 1201 UTC. Fields shown are (a) reflectivity (dBZ) and (b) radial velocity (kts). The 0.5 degree elevation angle is shown with range rings at 50 km intervals.

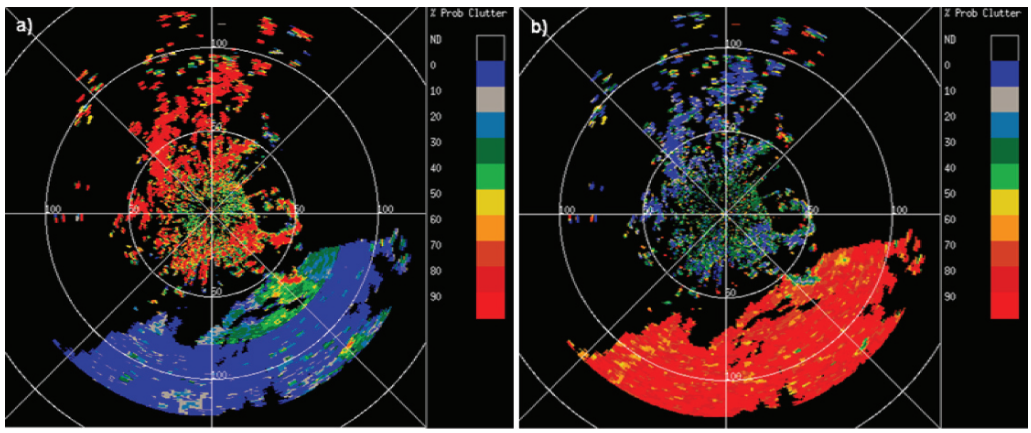


Fig. 17.14 Same as Fig. 17.13 except that the fields shown are the output from the (a) REC-APDA and the (b) REC-PDA. Warmest (red) colors indicate a likelihood approaching 1 (or 100%).

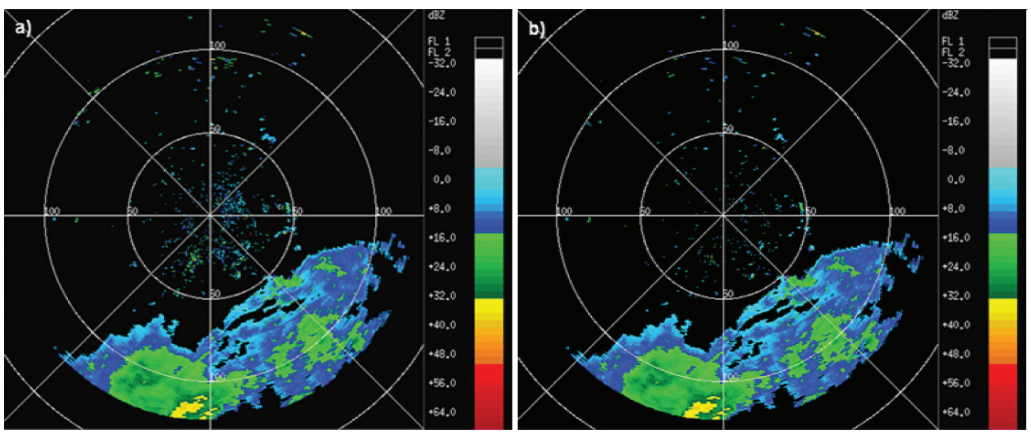


Fig. 17.15 Same as Fig. 17.13 except that the fields shown are the output from the (a) HSR with only the REC-APDA used to remove ground clutter contamination and the (b) HSR with both the REC-APDA and the REC-PDA used to remove ground clutter contamination.

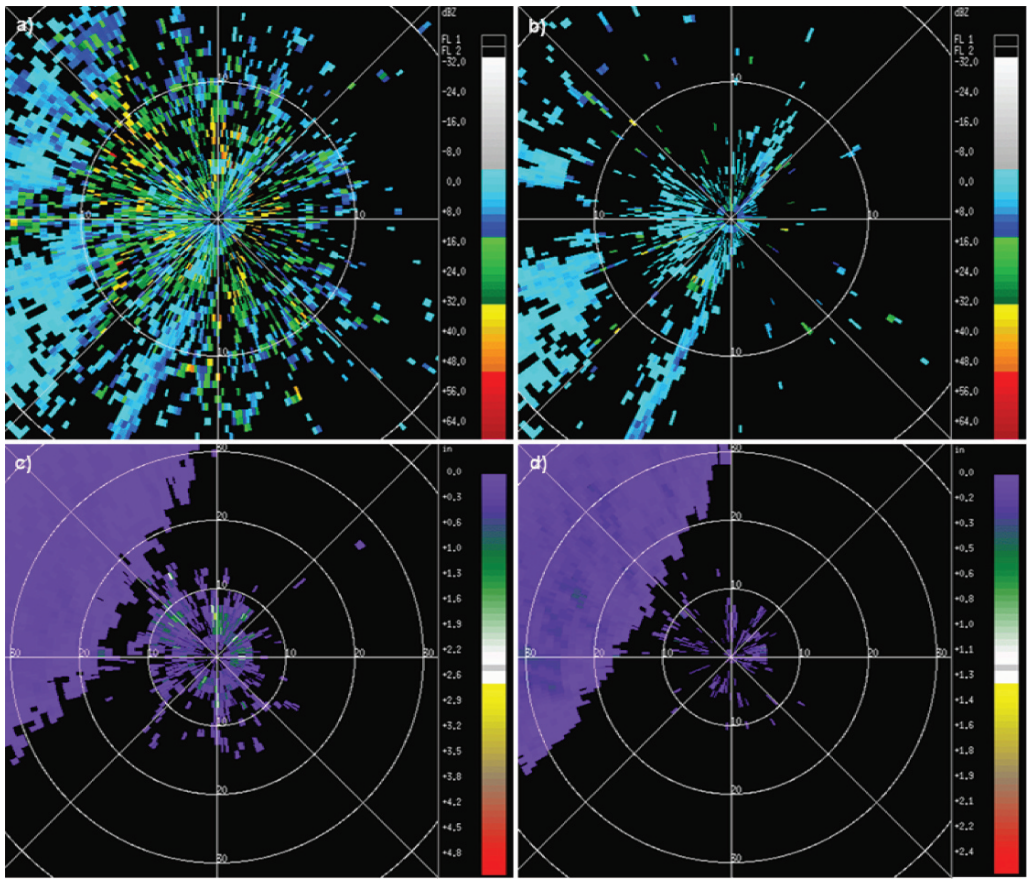


Fig. 17.16 A comparison of the HSR and the radar-derived rainfall amounts after 4 h of accumulation for an approaching squall line near the Chicago, IL (KLOT) WSR-88D. Fields shown are (a) the HSR (dBZ) when only the REC-APDA removes ground clutter contamination, (b) the HSR when both the REC-APDA and REC-PDA remove ground clutter contamination, (c) the 4-h accumulated rainfall (in) using the HSR illustrated in (a) and (d) the 4-h accumulated rainfall (in) using the HSR illustrated in (b). Note that the color table in (c) extends to twice the range of the color table in (d) due to the spurious high values near the radar. Range rings are at 10 km intervals in all panels, but (c) and (d) extend to a farther range than (a) and (b).

17.3.6 REC Sea Clutter Detection Algorithm (REC-SCDA)

Sea clutter poses a particularly difficult echo classification problem because, unlike ground clutter, sea clutter return typically has non-zero radial velocity. Likewise, discriminating precipitation return from sea clutter return can be difficult and, for this reason, the REC-SCDA is only applied over ocean regions to minimize the incorrect removal of precipitation return. A terrain mask is computed using a detailed terrain map to calculate which radar range gates are over water and which are over land. The REC-APDA is applied over both land and sea surfaces because second trip ground clutter return has been observed to occur over the sea under strong AP conditions.

Further, sea clutter characteristics can change depending on the sea state which, in turn, affects the performance of the REC-SCDA. At times, the feature field characteristics within sea clutter (such as reflectivity texture) resemble ground clutter (i.e., texture values are high) and at other times, feature fields more closely resemble precipitation (i.e., texture values are low). Because retaining precipitation echo is more important than removing all sea clutter return, the REC-SCDA membership functions are tuned for

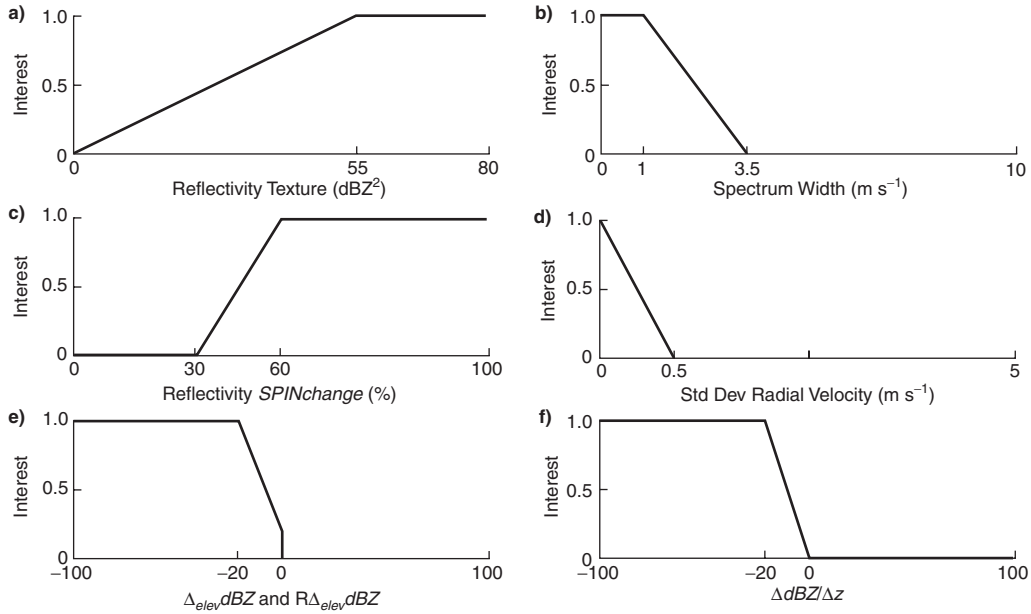


Fig. 17.17 Membership functions for each of the feature fields within the REC-SCDA. Membership functions shown are for (a) reflectivity texture (dBZ^2), (b) spectrum width (m s^{-1}), (c) reflectivity *SPINchange* (%), (d) standard deviation of the radial velocity (m s^{-1}), (e) the change in reflectivity between elevation angles ($\Delta_{elev}dBZ$) and the range-corrected change in reflectivity between elevation angles ($R\Delta_{elev}dBZ$) and (f) the vertical gradient of reflectivity ($\frac{\Delta dBZ}{\Delta z}$).

times when the sea clutter characteristics most closely resemble ground clutter.

In the United Arab Emirates (UAE), both AP ground clutter and sea clutter return can be extensive and in excess of 40–50 dBZ due to the presence of extreme moisture gradients near the top of the planetary boundary layer. The REC-APDA and the REC-SCDA were deployed at the coastal radars (NCAR Research Applications Program 2003). Each radar has slightly different system characteristics and, therefore, required that membership functions be tuned individually. As a result, membership functions for a particular feature field are not necessarily uniform for all radars.

The REC-SCDA uses the following feature fields: the reflectivity texture, the spectrum width, the reflectivity *SPINchange* using a reflectivity threshold of 2 dBZ, the standard deviation of the radial velocity, change in reflectivity between elevation angles ($\Delta_{elev}dBZ$), the range-weighted change in reflectivity between elevation angles ($R\Delta_{elev}dBZ$) and the vertical gradient of reflectivity ($\frac{\Delta dBZ}{\Delta z}$). As implemented within the Dalma radar, the membership functions for each feature field are shown in Fig. 17.17. In the computation of the weighted mean sum, a weight of unity is applied to the interest fields of spectrum width, *SPINchange*, $\Delta_{elev}dBZ$, $R\Delta_{elev}dBZ$ and $\frac{\Delta dBZ}{\Delta z}$ while a weight of 0.5 is applied to the reflectivity texture and the standard deviation of the radial velocity.

Figure 17.18 shows the (a) original and (b) thresholded reflectivity from the UAE Dalma radar. This case was selected to show the good discrimination achieved between the precipitation return and the sea clutter return. While some sea clutter is retained in (b), the overall quality of the thresholded data is high, with only a few gates of precipitation removed incorrectly.

17.3.7 Summary

High quality radar moment data can be identified through the use of a fuzzy logic consensus methodology called the Radar Echo Classifier (REC). The REC consists of several modular algorithms that are designed to detect particular types of scatterers. Once

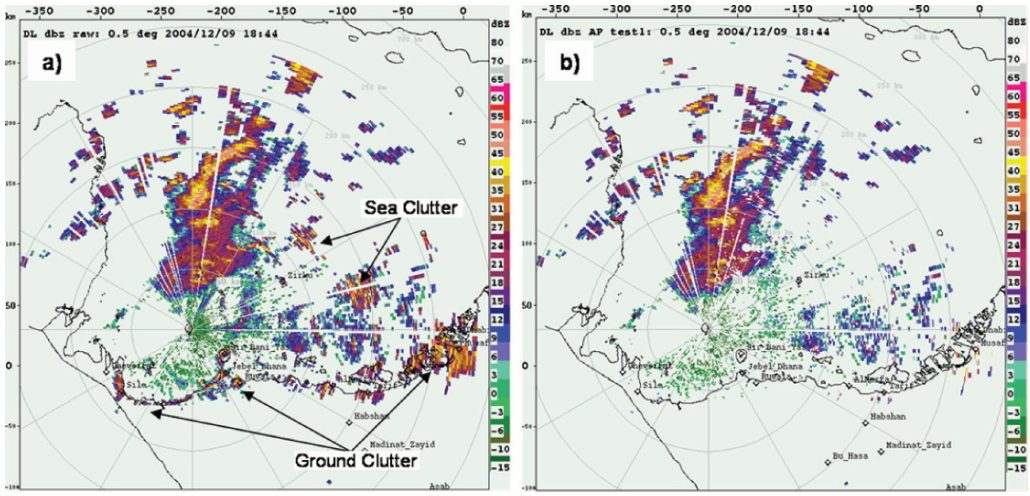


Fig. 17.18 The reflectivity (dBZ) field is shown from the UAE Dalma radar on 19 December 2004. The field is shown (a) in its original state with precipitation, ground clutter and sea clutter return and (b) after application of the REC-APDA and the REC-SCDA to remove the ground and sea clutter returns.

an undesirable scatterer is identified, it can be removed from the radar moment data. Likewise, the identification of desirable scatterers allows them to be retained. Before implementation of the REC-APDA, the WSR-88D radar network had no automated means of identifying AP ground clutter. Through examination of REC-APDA output, the radar operators can see when AP conditions are creating additional ground clutter and take steps to apply additional clutter filters to remove the contaminants. In the future, fuzzy logic techniques for clutter detection and discrimination from precipitation will occur within the WSR-88D Radar Data Acquisition (RDA; Dixon et al. 2007) such that human intervention will not be required and improved data quality will be achieved. For the UAE radar network, the REC-APDA and the REC-SCDA are applied automatically without human assistance.

17.4 Forecasting Clear-Air Turbulence

Pilots' ability to avoid turbulence during flight affects the comfort and safety of the millions of people who fly commercial airlines and other aircraft every year. Of all weather-related commercial aircraft incidents, 65% can be attributed to turbulence encounters, and major carriers estimate that they receive hundreds of injury claims that cost tens of millions of dollars per year (Sharman et al. 2006). At upper levels, clear-air turbulence (CAT) is particularly hard to avoid because it is invisible to both the eye and traditional remote sensing techniques. To plan flight paths that avoid turbulence, air traffic controllers, airline flight dispatchers, and flight crews must know where CAT pockets are likely to develop. Encountering unexpected turbulence is a serious safety risk, but diverting or canceling flights to avoid regions of suspected turbulence can cause added fuel expenditures or significant air traffic disruptions, so it is important that decision makers have information that is as accurate and reliable as possible.

Currently, the National Oceanic and Atmospheric Administration's (NOAA's) Aviation Weather Center (AWC) disseminates advisories called Airmen's Meteorological Information (AIRMETs) and Significant Meteorological Information (SIGMETs) for turbulence and other hazardous weather conditions. AIRMETs, issued regularly on a six-hour schedule, warn of moderate turbulence of any kind, surface winds greater than 30 knots, or low-level wind shear. Turbulence SIGMETs, valid for up to four hours, are issued when conditions indicate severe to extreme clear-air or other turbulence not related to a thunderstorm. Both types of advisories cover an area of at least 3,000 square miles using flight waypoints, airports and known geographical features to define a polygon

demarcating the advisory area boundary (Federal Aviation Administration 2008).

While AIRMETs and SIGMETs have long provided vital information on hazardous conditions for pilots and flight planners, the products have limited utility due to their constrained resolution in time and space and limited information on the type and intensity of turbulence. Less than 1% of the atmosphere is thought to contain moderate or greater turbulence at any one time (Frehlich and Sharman 2004), and frequently only a small fraction of a 3,000 square mile or larger advisory area may contain hazardous conditions. Moreover, the locations of those areas may change over the 4 or 6 h period for which an advisory is valid, both geographically and in altitude. While pilots distinguish between light, moderate, severe and extreme turbulence intensities in their in-flight reports, AIRMETs and SIGMETs identify only moderate or severe-to-extreme intensities. In addition, SIGMETs warn pilots specifically of CAT only if it is severe or extreme, but not if the CAT is moderate, a level of turbulence that pilots still prefer to avoid.

To address the challenges of turbulence forecasting and improve upon the spatial, temporal and intensity resolution of AIRMETs and SIGMETs, an automated CAT forecasting decision support product, Graphical Turbulence Guidance (GTG), was developed at NCAR and NOAA under direction and funding from the Federal Aviation Administration (FAA), and currently runs operationally at the AWC. The GTG algorithm integrates qualitative and quantitative reasoning about atmospheric conditions and observations using fuzzy logic to produce a CAT forecast every hour, for every flight level at a 20 km horizontal resolution. An example of the graphical forecast output as displayed on the AWC's Aviation Digital Data Service (ADDS; http://adds.aviationweather.gov) is shown in Fig. 17.19. The purpose of this section is to describe how GTG was developed using a fuzzy logic methodology.

17.4.1 Challenges Affecting Algorithm Development

One might assume that the best way to forecast turbulence would be to derive it from high-resolution numerical weather prediction (NWP) model wind fields. NWP models initialize grids using current observations, and simulate dynamical and physical processes to produce a forecast of the weather. The resolution, or spatial scale, of the model determines what weather processes it can predict accurately. For instance, a model with a horizontal resolution of 20 km cannot identify exactly where a thunderstorm cell, typically less than 10 km across, might develop; in NWP terms, this model cannot *resolve* thunderstorm-scale weather events.

Turbulence is an even larger (or smaller!) problem to resolve. Turbulence exhibits structure at many scales, all of which trade energy with one another in complicated ways. Forecasting the individual gusts that can bounce an aircraft up and down would require high-resolution observations and accurate fine-scale models (perhaps 25 m resolution) that are well beyond the capability of current technology. And even if modern computing power were sufficient, the nonlinear dynamics of the atmosphere would sharply limit the time over which such small-scale motions could be accurately predicted. However, turbulence can be characterized as a stochastic (random) cascade of energy from large-scale forcing to small-scale eddies, and estimating the average magnitude of the aircraft-scale eddies is more tractable. Still, currently operational NWP models employ horizontal resolutions that are on the order of 10–100 km, or two to three orders of magnitude larger than the scales of turbulence that affect aircraft. While models exist that directly simulate aircraft-scale turbulence, they can only be run retrospectively on large multiprocessor systems for research purposes. To extend these models to the entire U.S. airspace would require several orders of magnitude more computing power, communications bandwidth, and data storage capacity than is available today or will be in the foreseeable future. These computational constraints necessarily affect the development of an operational forecasting system such as GTG: the product must be able to run in a timely manner using available operational computing and storage resources.

17.4.2 Links to Large-Scale Atmospheric Conditions

Since computational resource requirements preclude direct predictions of turbulence, coarser operational

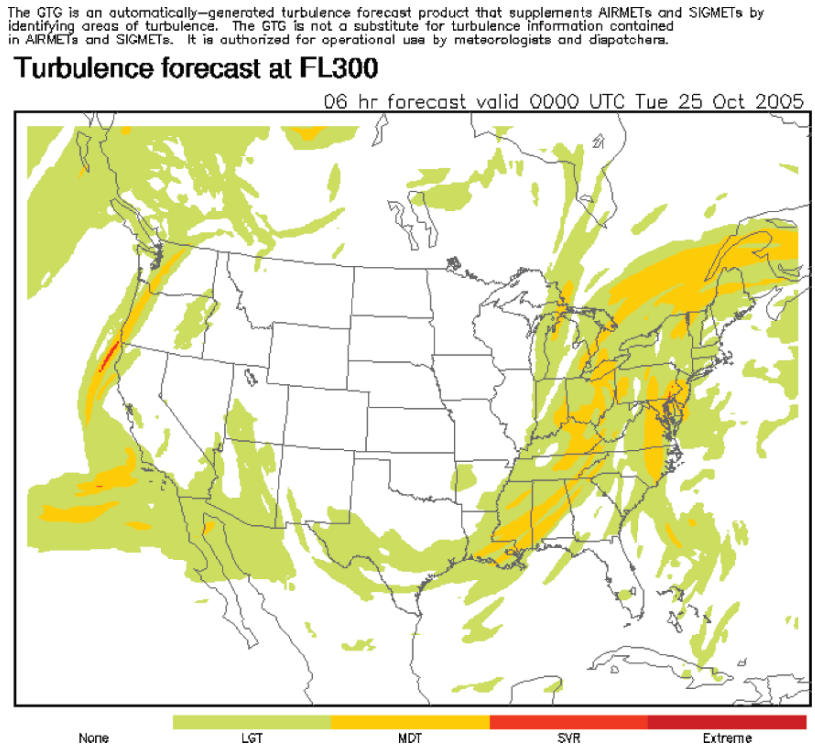


Fig. 17.19 Sample GTG 6-h forecast of CAT potential for 0000 UTC 25 October 2005 at FL300 (30,000 ft) as provided on the operational ADDS website (http://adds.aviationweather.gov/turbulence). The colorscale for the turbulence intensity contours is provided at the bottom of the image and ranges from "none" (white) to "extreme" (dark red). For the time shown, no turbulence is predicted over most of the western and midwestern U.S., with some light to moderate turbulence regions expected on both the east and west coasts and a narrow region of moderate to severe turbulence off the northern California coast. Source: Figure 1 from Sharman et al. (2006).

NWP model output must be used instead and analyzed for atmospheric conditions that may be conducive to turbulence. This approach should be physically sound, since research has shown that the energy associated with turbulent eddies at aircraft scales cascades down from larger scales of atmospheric motion (e.g., Nastrom and Gage 1985; Lindborg 1999, 2006; Koshyk and Hamilton 2001). Through years of producing manual forecasts, forecasters developed "rules of thumb" from their experience about what atmospheric conditions typically indicated turbulence. These rules of thumb were an attempt to link the large-scale meteorological data that was available and the micro-scale CAT. Researchers later quantified some of these rules into CAT *diagnostics* that are computed directly from the NWP model output.

For instance, a major cause of CAT is thought to be Kelvin-Helmholtz instability (e.g., Dutton and Panofsky 1970), that can occur in areas of low Richardson number (Ri, the ratio of static stability to wind shear). Thus, one CAT diagnostic derives Ri from the NWP model output fields of temperature (to derive static stability) and wind (to derive wind shear). Other diagnostics can be similarly derived from the NWP model output variables. There are 40 different CAT diagnostics available for use in GTG, each linking a large-scale atmospheric condition to small-scale turbulence. Some of these are described in Sharman et al. (2006). Their predictive powers vary, depending on how directly the large-scale condition is linked to turbulence, and some of them are correlated. Currently, GTG uses sets of about ten of the available diagnostics in each of its forecast domains.

Airline pilots provide occasional reports of conditions encountered during flight, called Pilot Reports (PIREPs) that may characterize turbulence as "light"

or "moderate to severe", for instance, at a specific time and location. These qualitative observations helped forecasters build the human knowledge base that resulted in the quantitative diagnostic equations available today. Now, researchers use PIREPs to gauge how well a diagnostic – or a combination of diagnostics – is currently predicting turbulence. Using PIREPs to evaluate the diagnostics' predictive skill as synoptic conditions change is integral to the GTG algorithm.

17.4.3 GTG Algorithm

The goal of GTG is to use NWP model output, PIREPs, and other observations of turbulence to supply reliable CAT forecasts in a system capable of running operationally within the computational limitations present at NOAA's AWC. The CAT forecast product is generated at the same frequency as the NWP model grids on which it is based, providing timely updates to aid users in planning flight paths to avoid turbulence. Therefore, the algorithm must be computationally efficient; in addition, it should combine multiple CAT diagnostics to capture all the conditions known to induce aircraft-scale turbulence. However, different sources of turbulence may predominate at different times, so it is also desirable to evaluate how well each diagnostic has performed recently based on comparisons with PIREPs and weight the diagnostic accordingly. Unfortunately, the relative infrequency of turbulence encounters and PIREPs means that not enough training data are available to produce stable performance by most machine learning algorithms.

Researchers found a solution in fuzzy consensus reasoning (see Chapter 6, Section 6.8). Every hour, CAT diagnostics are calculated from the NWP model analysis, or zero hour forecast, at every model grid point, and their values are compared to any available PIREPs from the corresponding time period and location. If a diagnostic value agrees well with available PIREPs (observations) of turbulence, the algorithm increases the weight of that diagnostic; otherwise, it lowers the weight. This process, which is described in detail below, is similar to how a human forecaster might reason about how different large-scale conditions are currently indicating turbulence. The forecast at 1, 2, 3, 6, 9 and 12 hours is produced by using the weighted sum of the diagnostic values on the forecast grids according to the fuzzy consensus reasoning equation (6.6). In this case, there are n diagnostics and x represents a physical location, or grid point, in the forecast area. Note that no de-fuzzification is needed; with fuzzy consensus reasoning, $\hat{u}(x)$ is the forecasted turbulence intensity at location x. Since the diagnostics are derived from NWP model output, which is assumed equally good everywhere, the data quality control "confidences" are all taken to be 1. The weight w_i represents the relative value of diagnostic f_i based on its recent performance evaluated over the forecast area; if PIREPs were more temporally and spatially dense, the weights could be computed separately for different regions. If the n weights are chosen so that they sum to 1, the GTG weighted sum equation can be simplified to:

$$\hat{u}(x) = \sum_{i=1}^{n} w_i \, f_i(x) \tag{17.10}$$

To determine the weights, the qualitative PIREPs must be compared to the quantitative diagnostic values. A pilot reports turbulence as null, light, moderate, severe or extreme turbulence. These sets have fuzzy boundaries due to the differences in pilot experience, subjective judgment, and the flight conditions and size of the airplane (atmospheric turbulence will generally affect a small plane more than a large plane). In addition, pilots sometimes report a turbulent event as "light to moderate", for example, translating into partial membership in both the light and moderate sets. In contrast, a turbulence diagnostic is a quantitative value in some continuous range. To scale raw turbulence diagnostic values to a consistent turbulence scale comparable to PIREPs, the values are mapped using thresholds based on long-term empirical studies: thresholds (T1, T2, T3, T4, T5) represent the medians of the raw diagnostic value corresponding to null, light, moderate, severe and extreme turbulence PIREPs, respectively (see Sharman et al. 2006). Using the thresholds, raw diagnostic values are remapped to the range 0 (null turbulence) to 1 (extreme turbulence), with diagnostic values falling between two thresholds scaled linearly between them as shown in Fig. 17.20. PIREPs are represented as discrete values according to the assignments 0 = null, 0.125 = null/light, 0.25 = light, 0.375 = light/moderate, 0.5 = moderate, 0.625 = moderate/severe, 0.75 = severe, 0.875 = severe/extreme, and 1 = extreme. Note that since the

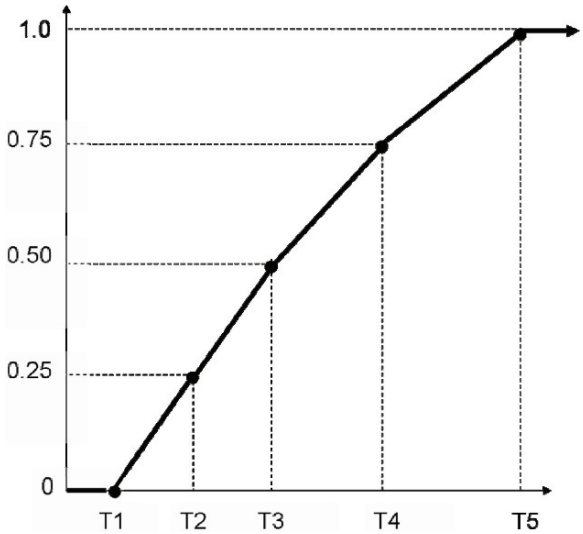


Fig. 17.20 Diagnostic mapping diagram. The raw values of the diagnostic are read along the abscissa, with specified thresholds T1 corresponding to the diagnostic value for null, T2 for light, T3 for moderate, T4 for severe, and T5 for extreme turbulence values. These are mapped to a 0–1 scale as indicated on the ordinate, with 0.25 being the light, 0.5 the moderate, 0.75 the severe, and 1.0 the extreme threshold. Note that raw diagnostic values < T1 are always mapped to null, and raw diagnostic values > T5 are always mapped to extreme. Source: Figure 3 from Sharman et al. (2006).

remapped diagnostic values all fall between 0 and 1, the consensus turbulence value provided by (17.10) will also be in this range.

While turbulence may be characterized by multiple categories, pilots tend to be most concerned with avoiding turbulence that is of moderate or greater (MOG) intensity. Light turbulence may not be enough of a safety or comfort hazard to warrant the cost of re-routing around it, and research has shown that "light" is the most subjective category of turbulence PIREPs, lessening researchers' confidence in using it. Therefore, the GTG algorithm is tuned to discriminate between MOG turbulence and less than MOG turbulence, termed Null turbulence for simplicity. Note that the MOG set encompasses the moderate, severe, and extreme sets, and light/moderate PIREPs have partial membership in the MOG set. The Null turbulence set is the complement of MOG, $\mathrm{MOG}^{\mathrm{C}}$, and includes only null (smooth or light turbulence) PIREPs. For simplicity, the partial membership of the light/moderate PIREPs is represented as binary membership value of 1 in the MOG set. Therefore, the MOG threshold is quantified as 0.375 on GTG's 0–1 turbulence scale.

Table 17.2 An example of a contingency table for a CAT diagnostic at one forecast time. PIREP observations are compared to each diagnostic's predictions of turbulence intensity at that location. Correct predictions are listed in the upper left (correct MOG predictions) and lower right (correct Null predictions). PODY and PODN can be calculated from the contingency table values; PODY, for example, is the number of correct MOG predictions (34) divided by the total number of MOG observations (34 + 15).

	MOG diagnostic prediction	Null diagnostic prediction
MOG observations	34	15
Null observations	12	50

If the PIREP and the remapped diagnostic value at a certain location x are both above or both below 0.375, they are considered in agreement, and the diagnostic has correctly classified turbulence at that location. For each NWP model analysis time, counts of agreement and disagreement with recent PIREPs are tallied in a contingency table for each diagnostic, an example of which is shown in Table 17.2. The Probability of Detection of a MOG turbulence event, POD-Yes (PODY), is the fraction of correct MOG classifications out of the set of all MOG PIREP observations (see Chapter 2). Likewise, POD-No (PODN) is the fraction of correct Null classifications out of the set of all Null PIREPs. From these PODY and PODN values, the diagnostic's True Skill Score (TSS) is calculated:

$$TSS = PODY + PODN - 1 \tag{17.11}$$

Each diagnostic is then evaluated according to its TSS and the fraction of MOG turbulence (f_{MOG}) it predicts over the forecast area (low levels of atmospheric turbulence are expected at any given time), resulting in a score, ϕ, defined by

$$\phi = \frac{TSS + 1.1}{1 + C\ (f_{MOG})^p} \tag{17.12}$$

Currently, GTG uses $C = 1$ and $p = 0.25$. The weight for the i^{th} diagnostic is then formed via:

$$w_i = \frac{\phi_i}{\sum_{m=1}^{n} \phi_m} \tag{17.13}$$

where n is the number of diagnostics; note that (17.13) ensures that $\sum_{i=1}^{n} w_i = 1$. The scaled diagnostics f_i are then combined using these weights to form the GTG consensus forecast according to (17.10), where x ranges over all positions in the 3-D forecast grid. The

same set of weights is applied to the scaled diagnostic values calculated from each NWP model forecast to produce the GTG forecasts. The GTG forecast at each grid point is between 0 and 1 and may be interpreted on the PIREP scale described above. These sets are represented as shaded contours in the GTG graphical output display shown in Fig. 17.19.

17.4.4 GTG Operational Details

The previous section outlined the GTG algorithm. To build the algorithm into an operational decision support product, the system has to coordinate receiving the NWP model data and observation data from different sources and producing graphical output for users, in addition to simply implementing the fuzzy logic algorithm. The schematic of the GTG forecasting system is shown in Fig. 17.21.

GTG currently uses the National Center for Environmental Prediction's (NCEP's) operational Rapid Update Cycle (RUC) weather model (Benjamin et al. 2004) output at 20 km resolution as a representation of large-scale atmospheric processes and the basis for computing the turbulence diagnostics. Pilot reports, obtained from some airlines and the NWS, and cloud-to-ground lightning flash data from the National Lightning Detection Network (NLDN; Murphy et al. 2006) are also used as observation data inputs. Lightning flashes are signs of thunderstorms; lighting data are used to filter out PIREPs near convection, since GTG is designed for forecasting only CAT and not convectively-induced turbulence (CIT). The fuzzy consensus reasoning takes place in the "GTG Nowcast and Forecast Generator" box. Every hour, the AWC receives RUC analysis time (zero hour "nowcast") and 1, 2, 3, 6, 9, and 12 h forecast files. Pilot reports and lightning flash data are updated every few minutes. The system looks at the time of the pilot report to determine whether to include it as an observation in the current hourly run of the algorithm. If there are fewer than 100 PIREPs in the applicable time window, as often happens at night, a default set of climatologically-derived weights is used for that hour's nowcast and forecasts in place of the weights determined by (17.13).

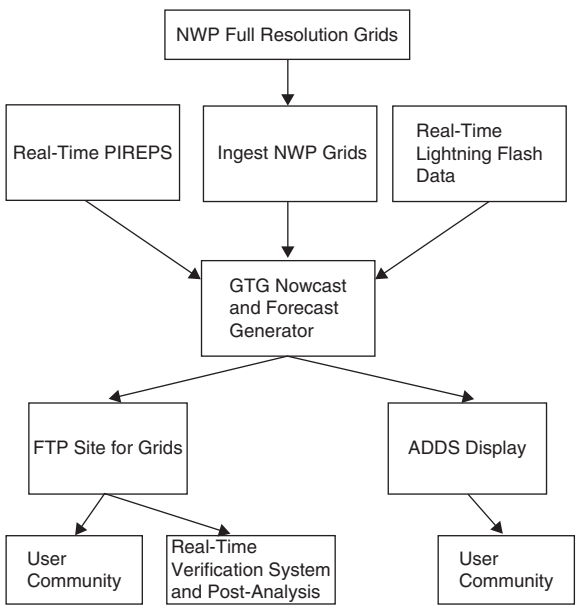


Fig. 17.21 The operational GTG forecasting system and its inputs and outputs Source: Figure provided by Robert Sharman.

Currently, GTG uses ten CAT diagnostics to make nowcasts and forecasts for 20,000 to 45,000 ft (upper levels), and nine CAT diagnostics for 10,000 to 20,000 ft (mid-levels), each with its own set of weights (Sharman et al. 2006). This distinction is made because different mechanisms are responsible for CAT at different atmospheric levels. While GTG is not the only turbulence forecasting system to use multiple diagnostics, it appears to be the first to combine them dynamically at forecast time.

17.4.5 GTG Performance

Turbulence forecast performance may be verified against PIREP observation data from the forecast valid time; a six-hour forecast at 12 UTC, for instance, has a valid time of 18 UTC and would be verified against observations from 18 UTC. Turbulence forecast accuracy may be measured by the Receiver Operating Characteristic (ROC) curve (e.g., Hanley and McNeil 1982; Marzban 2004; see also Chapter 2), which plots PODY against PODN as the MOG identification threshold is varied between 0 and 1. Higher PODY-PODN combinations over the range of thresholds, which produce a larger area under the ROC curve, imply greater skill in distinguishing MOG and Null turbulence. ROC curves for GTG and numerous

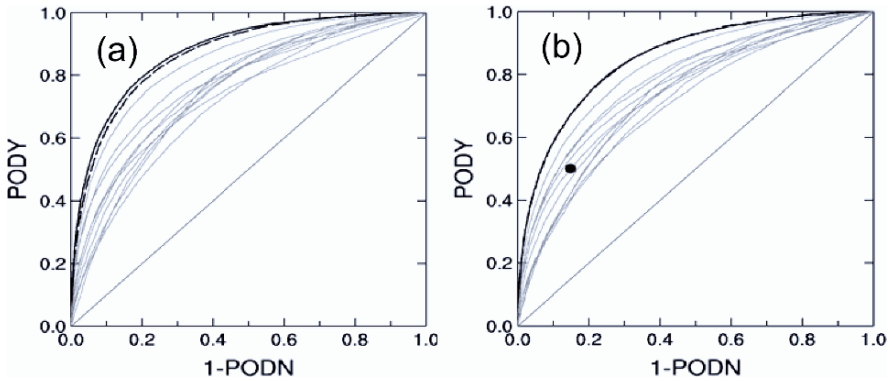


Fig. 17.22 Individual diagnostics and GTG PODY-PODN performance statistics (individual diagnostics as thin grey, GTG combination as heavy black solid and GTG combination using climatological weights as heavy black dashed ROC curves). These statistics were derived from one year (2003) of (a) 1800 UTC analyses (zero-hour forecasts) using 37,878 PIREPs and (b) 1800 UTC 6-h forecasts (valid 0000 UTC) using 49,703 PIREPs, for upper levels (FL200-FL460). For comparison, also shown is the diagonal "no skill" line, and the 2003 average AIRMET performance (with amendments) at upper levels for times around 2100 UTC (black dot). Source: Figure 5 from Sharman et al. (2006).

other turbulence diagnostics are shown in Fig. 17.22. The consensus fuzzy reasoning algorithm used by GTG produces a more accurate CAT forecast than does any single CAT diagnostic or the operational AIRMET.

17.4.6 Future Algorithm Development

Recently, objective turbulence observation data – termed *in situ* data – has become available through an effort by the FAA, some of the major airlines, and NCAR (Cornman et al. 1995, 2004). *In situ* measurements are obtained from onboard avionics data and recorded by special software on commercial aircraft at every minute during flight. The turbulence metric, which is computed and recorded at one-minute intervals, is the eddy dissipation rate (EDR), a quantitative measure of atmospheric turbulence intensity. These automated reports do not have the human subjectivity, aircraft-dependence and reporting delay issues of PIREPs (Schwartz 1996); moreover, they are produced routinely in flight regardless of whether there is turbulence or not.

The availability of these new data will allow researchers to reassess their choice of turbulence forecasting algorithms, and researchers are continuing to look to the AI community for efficient and effective approaches. The first step is to reassess CAT diagnostics' forecasting accuracies with this more objective observation data. For example, CAT climatologies suggest that the skill of various turbulence diagnostics vary regionally and seasonally. Researchers are exploring the topic of feature subset selection and employing searches to find well-performing subsets of diagnostics for different regions or synoptic conditions, with a goal of using fuzzy logic and other machine learning approaches to develop the next generation of GTG.

17.4.7 Summary

Adaptive fuzzy consensus reasoning provides a successful approach to clear-air turbulence forecasting and is currently being used operationally in the Graphical Turbulence Guidance (GTG) system. GTG utilizes atmospheric forecast information on a scale much coarser than the aircraft-scale turbulence, along with sparse, subjective reports of turbulence by pilots. The algorithm uses recent PIREPs to adapt its fuzzy consensus weights according to the varying forecasting performances of the model-derived turbulence diagnostics, which change in their utility over time as the mechanisms producing turbulence vary. Turbulence forecasting is a good example of a domain from a field of environmental science often dominated by explicit simulations. However, until computational resources to routinely model turbulence at aircraft scales are available, GTG's fuzzy logic combinations of turbulence diagnostics supplied

from available NWP model output will provide a viable and successful alternative.

17.5 Conclusion

This chapter has built on the introduction provided in Chapter 6 by describing in detail three fuzzy logic algorithms developed at the National Center for Atmospheric Research. All were designed to mimic a human expert's approach to a complex problem in meteorology. The NCAR Improved Moments Algorithm (NIMA) analyzes Doppler spectra to remove artifacts and provide better estimates of the spectral moments (reflectivity, velocity, and spectrum width), along with "confidence" values that represents their quality. It utilizes image processing techniques in a multi-stage, iterative approach. The Radar Echo Classifier (REC) provides an automated method for quality controlling operational Doppler radar data through the detection of anomalous propagation (AP) ground clutter, precipitation return and sea clutter contamination. It works by relating local features within the radar moment data to those features that characterize radar return from particular types of scatterers and then combining the results in a weighted consensus. The development of the algorithm is informed by an analysis of empirical data. Finally, the Graphical Turbulence Guidance (GTG) algorithm generates clear-air turbulence forecasts from numerical weather prediction model data. It utilizes a number of model-derived diagnostics that relate large-scale atmospheric features to the smaller-scale air motions that can shake an aircraft, dynamically weighting them in a consensus combination based on their recent performance as measured against pilot reports of turbulence. While these three applications only begin to illustrate the many ways in which fuzzy logic can be used in the environmental sciences, it is hoped that they have served as concrete examples that will stimulate the reader's imagination.

Acknowledgements Some of the research described in this chapter was performed in response to requirements and funding by the Federal Aviation Administration (FAA). The views expressed are those of the authors and do not necessarily represent the official policy or position of the FAA.

The NIMA algorithm was developed by Larry Cornman, Kent Goodrich, and Cory Morse. Cory Morse supplied the three figures used to explain and illustrate NIMA, and she and Kent Goodrich kindly provided helpful feedback on its description. The NOAA Radar Operations Center and the United Arab Emirates Rainfall Enhancement program provided funding for the Radar Echo Classifier. Robert Sharman led the development of GTG, and he provided useful feedback on its exposition.

References

Albo, D. (1994). Microburst detection using fuzzy logic. *Terminal area surveillance system program project report* (49 pp.). Washington, DC: Federal Aviation Administration. (Available from the author at NCAR, P. O. Box 3000, Boulder, CO 80307)

Albo, D. (1996). Enhancements to the microburst automatic detection algorithm. *Terminal area surveillance system program project report* (47 pp.). Washington, DC: Federal Aviation Administration. (Available from the author at NCAR, P. O. Box 3000, Boulder, CO 80307)

Barthazy, E., Goeke, S., Vivekanandan, J., & Ellis, S. M. (2001). Detection of snow and ice crystals using polarization radar measurements: Comparison between ground-based in situ and S-Pol observations. *Atmospheric Research*, 59–60, 137–162.

Benjamin, S. G., Grell, G. A., Brown, J. M., Sminova, T. G., & Bleck, R. (2004). Mesoscale weather prediction with the RUC hybrid isentropic-terrain-following coordinate model. *Monthly Weather Review*, 132, 473–494.

Bernstein, B. C., McDonough, F., Politovich, M. K., Brown, B. G., Ratvasky, T. P., Miller, D. R., Wolff, C. A., & Cunning, G. (2005). Current Icing Potential: algorithm description and comparison with aircraft observations. *Journal of Applied Meteorology*, 44, 969–986.

Carter, D. A., Gage, K. S., Ecklund, W. L., Angevine, W. M., Johnston, P. E., Riddle, A. C, Wilson, J., & Williams, C. R. (1995). Developments in UHF lower tropospheric wind profiling at NOAA's Aeronomy Laboratory. *Radio Science*, 30, 977–1001.

Cohn, S. A., Goodrich, R. K., Morse, C. S., Karplus, E., Mueller, S., Cornman, L. B., & Weekley, R. A. (2001). Radial velocity and wind measurements with NIMA: Comparisons with human estimation and aircraft measurements. *Journal of Applied Meteorology*, 40, 704–719.

Cornelius, R., Gagnon, R., & Pratte, F. (1995). Optimization of WSR-88D clutter processing and AP clutter mitigation. *Final Report* (182 pp.). Norman, OK: National Oceanic and Atmospheric Administration WSR-88D Operations Support Facility (OSF).

Cornman, L. B., Morse, C. S., & Cunning, G. (1995). Real-time estimation of atmospheric turbulence severity from in-situ aircraft measurements. *Journal of Aircraft*, 32, 171–177.

Cornman, L. B., Goodrich, R. K., Morse, C. S., & Ecklund, W. L. (1998). A fuzzy logic method for improved moment estimation from Doppler spectra. *Journal of Atmospheric and Oceanic Technology*, 15, 1287–1305.

Cornman, L. B., Meymaris, G., & Limber, M. (2004). An update on the FAA Aviation Weather Research Program's in situ turbulence measurement and reporting system. *Preprints,*

11th Conference on Aviation, Range, and Aerospace Meteorology. October 4–8, 2004. Hyannis, MA/Boston: American Meteorological Society. Paper 4.3. (Online at http://www.ametsoc.org)

Crum, T. D., & Alberty, R. L. (1993). The WSR-88D and the WSR-88D operational support facility. *Bulletin of the American Meteorological Society*, 74, 1669–1687.

Crum, T. D., Saffle, R. E., & Wilson, J. W. (1998). An update on the NEXRAD program and future WSR-88D support to operations. *Weather Forecasting*, 13, 253–262.

Davis, C., Brown, B., & Bullock, R. (2006). Object-based verification of precipitation forecasts. Part I: Methodology and application to mesoscale rain areas. *Monthly Weather Review*, 134, 1772–1784.

Delanoy, R. L., & Troxel, S. W. (1993). Machine intelligence gust front detection. *Lincoln Laboratory Journal*, 6, 187–211.

Dixon, M., Hubbert, J. C., Ellis, S., Kessinger, C., & Meymaris, G. (2007). Intelligent mitigation of normal and anomalous propagation clutter. *33rd International Conference on Radar Meteorology*. August 6–10, 2007. Cairns, Australia/ Boston: American Meteorological Society. Paper 8B.7. (Online at http://www.ametsoc.org)

Donaldson, R. J., Dyer, R. M., & Kraus, M. J. (1975). An objective evaluator of techniques for predicting severe weather events. *Preprints, 9th Conf. on Severe Local Storms* (pp. 321–326). October 21–23. Norman, OK/ Boston: American Meteorological Society.

Doviak, R. J., & Zrnić, D. S. (1993). *Doppler radar and weather observations* (2nd ed., 562 pp.). San Diego, CA: Academic Press.

Dutton, J., & Panofsky, H. (1970). Clear air turbulence: A mystery may be unfolding. *Science*, 167, 937–944.

Federal Aviation Administration (2008). Aeronautical Information Manual: Official Guide to Basic Flight Information and ATC Procedures. (Online at http://www.faa.gov)

Frehlich, R., & Sharman, R. (2004). Estimates of turbulence from numerical weather prediction model output with applications to turbulence diagnosis and data assimilation. *Monthly Weather Review*, 132, 2308–2324.

Fujita, T. T., & Byers, H. R. (1977). Spearhead echo and downbursts in the crash of an airliner. *Monthly Weather Review*, 105, 129–146.

Fujita, T. T., & Caracena, F. (1977). An analysis of three weather-related aircraft accidents. *Bulletin of the American Meteorological Society*, 58, 1164–1181.

Gerding, S., & Myers, B. (2003). Adaptive data fusion of meteorological forecast modules. *Preprints, 3rd Conference on Artificial Intelligence Applications to the Environmental Sciences*. February 9–13, 2003. Long Beach, CA, Boston: American Meteorological Society. Paper 4.8. (Online at http://www.ametsoc.org)

Goodrich, R. K, Morse, C. S., Cornman, L. B., & Cohn, S. A. (2002). A horizontal wind and wind confidence algorithm for Doppler wind profilers. *Journal of Atmospheric and Oceanic Technology*, 19, 257–273.

Hanley, J. A., & McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143, 29–36.

Herzegh, P. H., Wiener, G. M., Bankert, R., Bateman, R., Chorbajian, B., & Tryhane, M. (2004). Automated analysis and forecast techniques for ceiling and visibility on the national scale. *Preprints, 11th Conference on Aviation, Range, and Aerospace Meteorology*. 4–8 October 2004. Hyannis, MA/Boston: American Meteorological Society. Paper 10.1. (Online at http://www.ametsoc.org)

Hildebrand, P. H., & Sekhon, R. S. (1974). Objective determination of the noise level in Doppler spectra. *Journal of Applied Meteorology*, 13, 808–811.

Hjelmfelt, M. (1988). Structure and life cycle of microburst outflows observed in Colorado. *Journal of Applied Meteorology*, 27, 900–927.

Keeler, R. J., Kessinger, C., Van Andel, J., & Ellis, S. (1999). AP clutter detection and mitigation: NEXRAD implementation plan. *Preprints, 29th Conference on Radar Meteorology* (pp. 580–581) 12–16 July 1999. Montreal, Quebec, Canada/Boston: American Meteorological Society.

Keeler, R. J., Lutz, J., & Vivekanandan, J. (2000). S-Pol: NCAR's polarimetric Doppler research radar. *Proceedings of the International Geoscience and Remote Sensing Symposium* (Vol. 4, pp. 1570–1573). July 24–28, 2000. Honolulu, HI: IEEE.

Kessinger, C., Ellis, S., & Van Andel, J. (2003). The radar echo classifier: A fuzzy logic algorithm for the WSR-88D. *Preprints, 3rd Conference on Artificial Applications to the Environmental Sciences*, 9–13 February 2003. Long Beach, CA/Boston: American Meteorological Society. Paper P1.6. (Online at http://www.ametsoc.org)

Koshyk, J. N., & Hamilton, K. (2001). The horizontal energy spectrum and spectral budget simulated by a high-resolution troposphere-stratosphere-mesosphere GCM. *Journal of Atmospheric Science*, 58, 329–348.

Lindborg, E. (1999). Can the atmospheric kinetic energy spectrum be explained by two-dimensional turbulence? *Journal of Fluid Mechanics*, 388, 259–288.

Lindborg, E. (2006). The energy cascade in a strongly stratified fluid. *Journal of Fluid Mechanics*, 550, 207–242.

Marzban, C. (2004). The ROC curve and the area under it as performance measures. *Weather Forecasting*, 19, 1106–1114.

McDonough, F., Bernstein, B. C., Politovich, M. K., & Wolff, C. A. (2004). The forecast icing potential (FIP) algorithm. *Preprints, 20th Int'l Conf. on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology*.. January 12–15, 2004. Seattle, WA/Boston: American Meteorological Society. Paper 3.2. (Online at http://www.ametsoc.org)

Merritt, M. W. (1991). Microburst divergence detection for Terminal Doppler Weather Radar (TDWR). *MIT Lincoln Laboratory Project Report ATC-181* (174 pp.). Lexington, MA: MIT Lincoln Laboratory.

Morse, C. S., Goodrich, R. K., & Cornman, L. B. (2002). The NIMA method for improved moment estimation from Doppler spectra. *Journal of Atmospheric and Oceanic Technology*, 19, 274–295.

Mueller, C., Saxen, T., Roberts, R., Wilson, J., Betancourt, T., Dettling, S., Oien, N., & Yee, J. (2003). NCAR auto-nowcast system. *Weather Forecasting*, 18, 545–561.

Murphy, M. J., Demetriades, N. W. S., Holle, R. L., & Cummins, K. L. (2006). Overview of capabilities and performance of the U.S. National Lightning Detection Network. *Preprints-CD, 12th Conf, on Aviation, Range, and Aerospace*

Meteorology. January 30–February 2, 2006. Atlanta, GA/ Boston: American Meteorological Society. Paper J2.5. (Online at http://www.ametsoc.org)

Nastrom, G. D., & Gage, K. S. (1985). A climatology of atmospheric wavenumber spectra of wind and temperature observed by commercial aircraft. *Journal of Atmospheric Science*, 42, 950–960.

NCAR Research Applications Program (2003). Rainfall enhancement and air chemistry studies. In *United Arab Emirates, 2001–2002 Final Report* (470 pp.). United Arab Emirates: Department for Water Resources Studies, The Office of His Highness the President. (pp. 20–31)

Oye, R., Mueller, C., & Smith, S. (1995). Software for radar translation, visualization, editing and interpolation. *Preprints, 27th Conference on Radar Meteorology* (pp. 359–361). October 9–13, 1995. Vail, CO/Boston: American Meteorological Society.

Pratte, F., Ecoff, D., Van Andel, J., & Keeler, R. J. (1997). AP clutter mitigation in the WSR-88D. *Preprints, 28th Conference on Radar Meteorology* (pp. 504–505). September 7–12, 1997. Austin, TX/Boston: American Meteorological Society.

Reed, J. R., & Cate, G. (2001). Status of WSR-88D open radar product generator development and deployment. *Preprints, 17th Int'l Conf. on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology* (pp. 102–104). October 24–29, 2005. Albuquerque, NM/ Boston: American Meteorological Society. (Online at http://www.ametsoc.org)

Roberts, R. D., Burgess, D., & Meister, M. (2006). Developing tools for nowcasting storm severity. *Weather Forecasting*, 21, 540–558.

Saffle, R. E., & Johnson, L. D. (1997). NEXRAD product improvement overview. *Preprints, 13th International Conference on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology* (pp. 223–227). February 2–7, 1997. Long Beach, CA/ Boston: American Meteorological Society.

Schwartz, B. (1996). The quantitative use of PIREPs in developing aviation weather guidance products. *Weather Forecasting*, 11, 372–384.

Sharman, R., Tebaldi, C., Wiener, G., & Wolff, J. (2006). An integrated approach to mid- and upper-level turbulence forecasting. *Weather Forecasting*, 21, 268–287.

Skolnick, M. I. (1980). *Introduction to radar systems* (2nd ed., 581 pp.). New York: McGraw-Hill.

Steiner, M., & Smith, J. A. (2002). Use of three-dimensional reflectivity structure for automated detection and removal of non-precipitating echoes in radar data. *Journal of Atmospheric and Oceanic Technology*, 19, 673–686.

Vivekanandan, J., Zrnic, D. S., Ellis, S. M., Oye, R., Ryzhkov, A.V., & J. Straka. (1999). Cloud microphysics retrieval using S-band dual-polarization radar measurements. *Bulletin of the American Meteorological Society*, 80, 381–388.

Weekley, R. A., Goodrich, R. K., & Cornman, L., (2003). Fuzzy image processing applied to time series. *Preprints, 3rd Conference on Artificial Intelligence Applications to the Environmental Sciences*. February 9–13, 2003. Long Beach, CA/Boston: American Meteorological Society. Paper 4.3. (Online at http://www.ametsoc.org)

Wilks, D. S. (1995). *Statistical methods in the atmospheric sciences* (464 pp.). San Diego, CA: Academic Press.

Environmental Optimization: Applications of Genetic Algorithms

18

Sue Ellen Haupt

18.1 The Nature of Optimization

The genetic algorithm (GA) has found wide acceptance in many fields, ranging from economics through engineering. In the environmental sciences, some disciplines are using GAs regularly as a tool to solve typical problems; while in other areas, they have hardly been assessed for use in research projects. The key to using GAs in environmental sciences is to pose the problem as one in optimization. Many problems are quite naturally optimization problems, such as the many uses of inverse models in the environmental sciences. Other problems can be manipulated into optimization form by careful definition of the cost function, so that even nonlinear differential equations can be approached using GAs (Karr et al. 2001; Haupt 2006). Although optimization is usually accomplished with more traditional techniques, using a genetic algorithm allows cost functions that are not necessarily differentiable or continuous (Marzban and Haupt 2005).

Chapter 5 of this volume provides an introduction to genetic algorithms and some basic examples that demonstrate their use. Chapter 14 describes a specific problem in optimization in which a GA proved useful – using field monitored contaminant concentration data coupled with a transport and dispersion model to back-calculate source and meteorological information. The purpose of this current chapter is to review some of the broad range of applications of genetic algorithms to environmental science problems, to present a few examples of how a GA might be applied to some problems, and to suggest how they may be useful in future research directions.

Section 18.2 reviews some of the applications of genetic programming, concentrating on applying genetic algorithms to problems in the environmental sciences. Then Sections 18.3 through 18.6 give specific examples and describe how GAs can help solve problems in different ways than more traditional techniques. Section 18.3 describes a method for building empirical models with a GA using the Lorenz attractor as an example. Section 18.4 provides an example of a GA used for assimilating time varying meteorological data into a dispersion model. A classification problem is discussed in Section 18.5 in the context of using a GA to train a neural network (NN). Section 18.6 shows how a GA can be used to find equilibrium solutions of a high order partial differential equation – finding solitary wave solutions that are difficult to identify with more standard techniques. A summary is provided in Section 18.7. The overall goal of this chapter is to give the reader some ideas on how a GA might be configured to address her/his own problems.

18.2 GA Applications – Natural Optimization

Better methods of optimization and search would be quite beneficial in the environmental sciences. As an example, many different problems involve fitting an inverse model to observed or model-produced data. Those data could be time series or observed environmental states. Sometimes general functional forms

Sue Ellen Haupt (✉)
Applied Research Laboratory and Department of Meteorology, The Pennsylvania State University, P.O. Box 30, State College, PA 16802, USA
Phone: 814/863-7135; fax: 814/865-3287;
email: haupts2@asme.org

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

are known or surmised from the data. Frequently, the goal is to fit model parameters to optimize the match between the constructed model and the data. Modelers often go the next step and use that newly constructed inverse model to make predictions. Hart et al. (1998) discuss the need for new tools involving artificial intelligence (AI) techniques, including genetic algorithms.

One example of fitting a model to observed data using a GA is reported by Mulligan and Brown (1998). They used a GA to estimate parameters to calibrate a water quality model. They used nonlinear regression to search for parameters that minimize the least square error between the best fit model and the data. They found that the GA works better than more traditional techniques plus noted the added advantage that the GA can provide information about the search space, enabling them to develop confidence regions and parameter correlations. Some other work related to water quality includes using GAs to determine flow routing parameters (Mohan and Loucks 1995), solving ground water management problems (McKinney and Lin 1993; Rogers and Dowla 1994; Ritzel et al. 1994), sizing distribution networks (Simpson et al. 1994), and calibrating parameters for an activated sludge system (Kim et al. 2002). In fact the use of GAs and more general evolutionary algorithms have become quite prevalent in the hydrological area and there are a plethora of applications in the literature. Recently, it has progressed into development of very efficient methods for finding Pareto fronts of optimal multiobjective criteria to best estimate parameters of hydrological models (Vrugt et al. 2003). A Pareto front represents multiple solutions that balance the competing cost functions.

Applications in managing groundwater supplies have been using AI and GAs. Peralta and collaborators combined GAs with neural networks and simulated annealing techniques. Aly and Peralta (1999a) used GAs to fit parameters of a model to optimize pumping locations and schedules for groundwater treatment. They then combined the GA with a neural network (NN) to model the complex response functions within the GA (Aly and Peralta 1999b). Shieh and Peralta (1997) combined Simulated Annealing (SA) and GAs to maximize efficiency. Fayad (2001) together with Peralta used a Pareto GA to sort optimal solutions for managing surface and groundwater supplies, including a fuzzy-penalty function while using an Artificial Neural Network (ANN) to model the complex aquifer systems in the groundwater system responses. Chan Hilton and Culver (2000) also used GAs to optimize groundwater remediation design.

An example of the successful application of a GA to classification and prediction was done by Sen and Oztopal (2001) who classified rainy day versus non-rainy day occurrences. They used the GA to estimate the parameters in a third order Markov model. McCullagh et al. (1999) combined GAs with neural networks for rainfall estimation. Garcia-Orellana et al. (2002) investigated various methods of using a neural net to classify clouds visible on Meteosat images. The GA method proved best given its ability to interact directly with the NN. A model to forecast summer rainfall over the Indian landmass was successfully developed by Kishtawal et al. (2003) using a GA and historical data.

GAs have also proven useful for interpreting remote sensing data. Babb et al. (1999) developed algorithms that included a GA to retrieve cloud microphysical parameters from Doppler radar spectra. Lakshmanan (2000) used hybrid GAs to tune fuzzy sets in order to identify bounded weak echo regions, which are radar return features associated with supercell thunderstorms. Gonzalez et al. (2002) were able to determine cloud optical thickness, effective droplet radius, and temperature of night-time large-scale stratiform clouds over the ocean by inverting an atmospheric radiative transfer model. Since the inversion displays multiple local minima, a GA was needed to produce a global minimum that agreed well with local *in situ* measurements. Finally, models of lightning current return strokes were built using a GA (Popov et al. 2000). That work optimized the parameters of analytic return stroke models.

One example from geophysics is determining the type of underground rock layers. Since it is not practical to take core samples of sufficient resolution to create good maps of the underground layers, modern techniques use seismic information or apply a current and measure the potential difference that gives a resistance. These various methods produce an underdetermined multimodal model of the Earth. Fitting model parameters to match the data is regarded as a highly nonlinear process. Genetic algorithms have been successful in finding realistic solutions for this inverse problem (Jervis and Stoffa 1993; Jervis et al. 1996; Sen and Stoffa 1992a, b, 1996; Chunduru et al. 1995,

1997; Boschetti et al. 1995, 1996, 1997; Porsani et al. 2000), among others. Minister et al. (1995) found that evolutionary programming is useful for locating the hypocenter of an earthquake, especially when combined with simulated annealing.

Another inverse problem is determining the source of air pollutants given what is known about monitored pollutants. Additional information includes the usual combination (percentages) of certain pollutants from different source regions and predominant wind patterns. The goal of the receptor inverse models is to target what regions, and even which sources contribute the most pollution to a given receptor region. This process involves an optimization. Cartwright and Harris (1993) suggest that a genetic algorithm may be a significant advance over other types of optimization models for this problem when there are many sources and many receptors. Loughlin et al. (2000) combined GAs with ozone chemistry air quality models to allocate control strategies to avoid exceeding air quality standards. A demonstration of this type of model appears in Chapter 14 of this volume.

Evolutionary methods have also found their way into oceanographic experimental design. Barth (1992) showed that a genetic algorithm is faster than simulated annealing and more accurate than a problem specific method for optimizing the design of an oceanographic experiment. Porto et al. (1995) found that an evolutionary programming strategy was more robust than traditional methods for locating an array of sensors in the ocean after they have drifted from their initial deployment location.

Charbonneau (1995) constructed three models using GAs in astrophysics: modeling the rotation curves of galaxies, extracting pulsation periods of Doppler velocities in spectral lines, and optimizing a model of hydrodynamic wind. Hassan and Crossley (2003) used GAs to configure constellations of satellites.

Finally, GAs have also been applied in the context of ecological modeling. Recknagel (2001) lists some general examples; Yin et al. (2002) describe using a GA-based multivariate linear regression model to construct quantitative structure-property relationships; and Fang et al. (2003) used a GA to retrieve leaf area index from Landsat images and a canopy radiative transfer model.

The studies cited here are merely some examples of how GAs have been useful in the environmental sciences. The literature on GA applications has been increasing rapidly in the past 10 years and it is impossible to give an exhaustive review. Instead we describe a few applications in more detail to give the reader some specific examples from a range of different problems.

18.3 Example 1 – Least Squares Empirical Models

18.3.1 Empirical Modeling

Numerical modeling of time dependent problems such as those found in geophysical fluid dynamics has traditionally involved using some type of time stepping scheme, either explicit or implicit, combined with known dynamics discretized from a partial differential equation. Sometimes, however, the details of the dynamics are not sufficiently known or we wish to develop a model that reproduces dynamic behavior without involving the details of the full physical equations. In these cases, it is common to construe the form of an empirical, or inverse, model and use either observed or model-produced data – together with statistical techniques – to fit the parameters of the model.

Inverse models are becoming increasingly common in science and engineering. Sometimes we have collected large amounts of data but have not developed adequate theories to explain the data. Other times, the theoretical models are so complex that it is extremely computer intensive to use them. In either case, it is often useful to begin with available data and fit a stochastic model that minimizes some mathematical normed quantity, that is, a cost. Our motivation here lies in trying to predict environmental variables. In recent years, many scientists have been using the theory of Markov processes combined with a least squares minimization technique to build stochastic models of environmental variables in atmospheric and oceanic science (Hasselmann 1988; Penland 1989; Penland and Ghil 1993). One example is predicting the time evolution of sea surface temperatures in the western Pacific Ocean as a model of the rises and falls of the El Niño/Southern Oscillation (ENSO) cycle. This problem proved challenging.

However, stochastic models have performed as well as the dynamical ones in predicting future ENSO cycles (Penland and Magorian 1993; Penland and Matrosova 1994). Kondrashov et al. (2005) extended the technique to a nonlinear form using multiple polynomial regression enabling capture of higher order moments of ENSO. Another application involves predicting climate. We now build very complex climate models that require huge amounts of computer time to run. There are occasions when it would be useful to predict the stochastic behavior of just a few of the key variables in a large atmospheric model without concern for the details of day-to-day weather. Branstator and Haupt (1998) fit linear empirical models of climate that reproduced a response to forcing better than a standard linearized quasi-geostrophic model. Would a different method of minimizing the function produce a better match to the environmental time series? This is an interesting question without a clear answer. Before answering it using large climate models, it is convenient to begin with simple low-dimensional models of analytical curves. We choose to use a genetic algorithm as the technique to find nonlinear solutions.

18.3.2 Linear Empirical Models

Linear empirical models are reasonably straightforward to produce from data using standard least squares inversion techniques. The simplest time-varying model is linear and can be written in the form:

$$\frac{d\mathbf{s}}{dt} = \mathbf{B}\mathbf{s} + \boldsymbol{\xi} \tag{18.1}$$

where: $\mathbf{s}$ is the N-dimensional state and can represent states such as velocities at various locations or the spectral coefficients of the velocity,

$\frac{d\mathbf{s}}{dt}$ is the time rate of change of the state,
$\mathbf{B}$ is a linear $N \times N$ tensor that relates the above two,
$\boldsymbol{\xi}$ is a vector of white noise.

The deterministic dynamics are contained in the matrix, $\mathbf{B}$. Nonlinearities are parameterized through corrections to $\mathbf{B}$ as well as within the noise, ξ. This simple linear form is easily fit using standard analytical techniques, that is, minimizing the square of the error between the model and the time averaged data tendencies. Specifically, we wish to minimize the normed error, requiring that:

$$E = \left\langle \left\{ \left(\frac{d\mathbf{s}}{dt} - \mathbf{B}\mathbf{s} \right)^p \right\} \right\rangle \text{ is minimized} \tag{18.2}$$

The curly braces represent a spatial averaging while the angle brackets represent an ensemble time average. The traditional solution to this problem involves finding where the derivative of E vanishes. p is any appropriate power norm that we choose. The least squares methods use $p = 2$, or, an L^2 norm. Upon solving the least squares problem ($p = 2$), we find:

$$\mathbf{B} = \ln\left(\frac{\boldsymbol{\Lambda}_\tau}{\boldsymbol{\Lambda}} \right) / \tau \tag{18.3}$$

$\boldsymbol{\Lambda} = \langle \mathbf{s}(t)\mathbf{s}(t) \rangle$ is the covariance tensor, averaged over time.
$\boldsymbol{\Lambda}_\tau = \langle \mathbf{s}(t+\tau)\mathbf{s}(t) \rangle$ is the lagged covariance tensor.
τ is the chosen lag time.

Note that the model depends on the value of the lag, τ, the amount of data used to build it, and the resolution of the data. For a more detailed discussion of the basis of this model and suggestions for how to formulate it, see the work of Penland (1989).

Here, we instead pose the optimization problems as finding the matrix $\mathbf{B}$ that minimizes the cost function defined by the error equation (18.2) and solve using a GA.

As an example, a time series is generated using $(X,\ Y,\ Z) = (\sin(t), \cos(t),\ t)$, with $t = [0,\ 10\pi]$ in increments of $\pi/50$ to form a spiral curve. The time evolution of this curve appears in Fig. 18.1a. Note that computation of the cost function requires a summation over 500 time increments. A continuous GA is applied to this curve with a population size of $N_{pop} = 100$, a mutation rate of $\mu = 0.2$, and run for 70 generations. Since any value of p can be used, we experimented a bit and found the best results for moderate p. The GA solution appears in Fig. 18.1b for p = 4. The general shape of the spiral curve is captured rather well. The bounds in X and Y are approximately correct, but the evolution in $Z = t$ is too slow. This aspect of the spiral is rather difficult to capture. In terms of dynamical systems, we were able to find the attractor, but not able to exactly model the evolution along it. For comparison, a standard least squares technique is used to solve the same problem and appears as Fig. 18.1c. We can

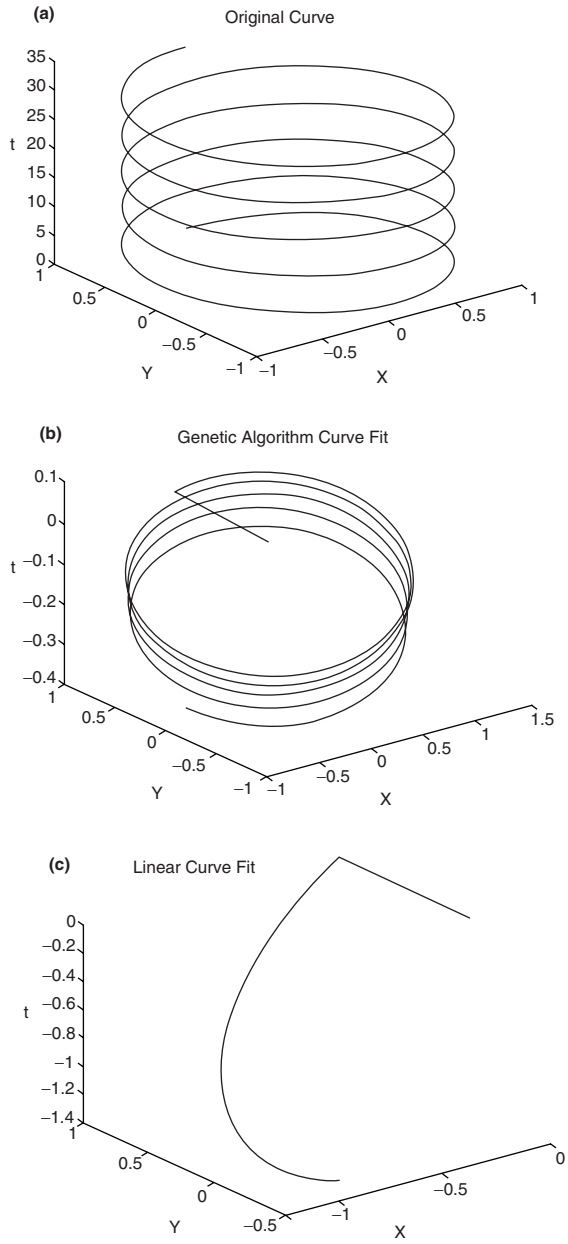


Fig. 18.1 The spiral curve. (a) Initial data created from $(X,\ Y,\ Z) = (\sin(t), \cos(t),\ t)$, with $t = [0,\ 10\pi]$ in increments of $\pi/50$ with Runge Kutta integration, (b) model fit with a GA, and p = 4, and (c) linear least squares model fit

see that the least squares method could not even come close to capturing the shape of the attractor. Of course, we can fine tune the least squares method by adding a noise term in the cost function. We can do that for the GA as well. The advantage of the GA is that it is easy to add complexity to the cost function. For this simple model, adding more variables or changing the power, p, adds nothing to the solution of the problem since it can be completely specified with the nine degrees of freedom in the matrix.

18.3.3 Nonlinear Empirical Models

When the dynamics are highly nonlinear, stochastic linear dynamics cannot be expected to reproduce all of the system's behavior (Penland 1989). It is thus convenient to formulate stochastic empirical models using nonlinear dynamics. Chapter 5 introduced the concept of a nonlinear empirical model in the context of reproducing the limit cycle of the predator-prey (Lotka Volterra) equations. As done there, we choose to concentrate on quadratic nonlinearity since: (1) It is the most reasonable to calculate. The higher the order of the problem, the more data is necessary to obtain a good fit. (2) A quadratic deterministic form is sufficient to produce the entire range of coupled dynamical behavior, such as limit cycles and chaotic motion. (3) The forward dynamical models used in geophysical fluid dynamics use a quadratic term to specify the nonlinear advection. Thus, it is consistent with the dynamical equations to expect quadratic nonlinearity.

Here, we formulate the quadratic empirical model as:

$$\frac{d\mathbf{s}}{dt} = \mathbf{s}\mathbf{C}\mathbf{s}^T + \mathbf{B}\mathbf{s} + \xi \tag{18.4}$$

The nonlinear interactions now occur explicitly through the nonlinear term involving the third order operator **C**. **B** is again an $N x N$ tensor that serves as the linear propagator, **C** is an $N\ x\ N\ x\ N$ third order tensor that gives the coefficients of the quadratic interactions. We wish to compute the tensors **B** and **C** so that the least square error of (18.4),

$$E = \left\langle \left(\frac{d\mathbf{s}}{dt} - \mathbf{B}\mathbf{s} - \mathbf{s}\mathbf{C}\mathbf{s}^T \right)^2 \right\rangle \text{ is minimized} \tag{18.5}$$

The angle brackets denote a time average. Since (18.4) is a quadratic generalization of (18.1), standard methods can be used for determining **B** and **C** (for instance see Menke 1984). Details are given in Haupt (2006). The problem is that the closed form solution requires

inverting a fourth order tensor, which is not trivial. Therefore, we choose to instead compute **C** by doing a best fit with a genetic algorithm. Symmetries can be invoked to reduce the dimensionality of the problem. The GA is initialized with each chromosome being a "guess" at the correct solution to **C**. The elements of **C** become the genes that are concatenated into chromosomes. The cost function for this problem is (18.5).

18.3.4 Lorenz Equations

As an example, we examine whether it is possible to capture the chaotic behavior of the three equation Lorenz system (Lorenz 1963), which can be written:

$$\begin{aligned} \frac{dx}{dt} &= -\sigma x + \sigma y \\ \frac{dy}{dt} &= \rho x - y - xz \\ \frac{dz}{dt} &= -bz + xy \end{aligned} \tag{18.6}$$

where x, y, z are the lowest order coefficients of a truncated series of atmospheric flow and we use parameters: $\sigma = 10, b = 8/3, \rho = 28$. These parameters produce a chaotic regime that results in a strange attractor, often referred to as the "butterfly" attractor. The equations (18.4) are integrated using a fourth order Runge-Kutta method to produce the data trajectory shown in Fig. 18.2a, in three dimensional phase space.

A nonlinear empirical model of these data is created using the techniques presented above. The parameters of the GA are an initial population of 500, working population of 100, crossover rate of 0.5, and mutation rate of 0.3 for a total of 200 generations. Taking into account symmetries for this problem results in 18 unique parameters in **C**. For this highly nonlinear regime, it is difficult to find a solution. Not every attempt converged to a small residual of the cost function. It required multiple attempts to produce the time trajectory shown in Fig. 18.2b. Although the match is not perfect, the general shape of the strange attractor is replicated.

For comparison, the solution due to a linear empirical model fit of (18.1) is shown in Fig. 18.2c, similar to that shown previously by Penland (1989). The linear model did not capture the shape of the attractor, but instead shows a decaying spiral behavior. Although the parameters were chosen to model the chaotic regime, the linear model can only capture a homoclinic orbit toward a stable fixed point.

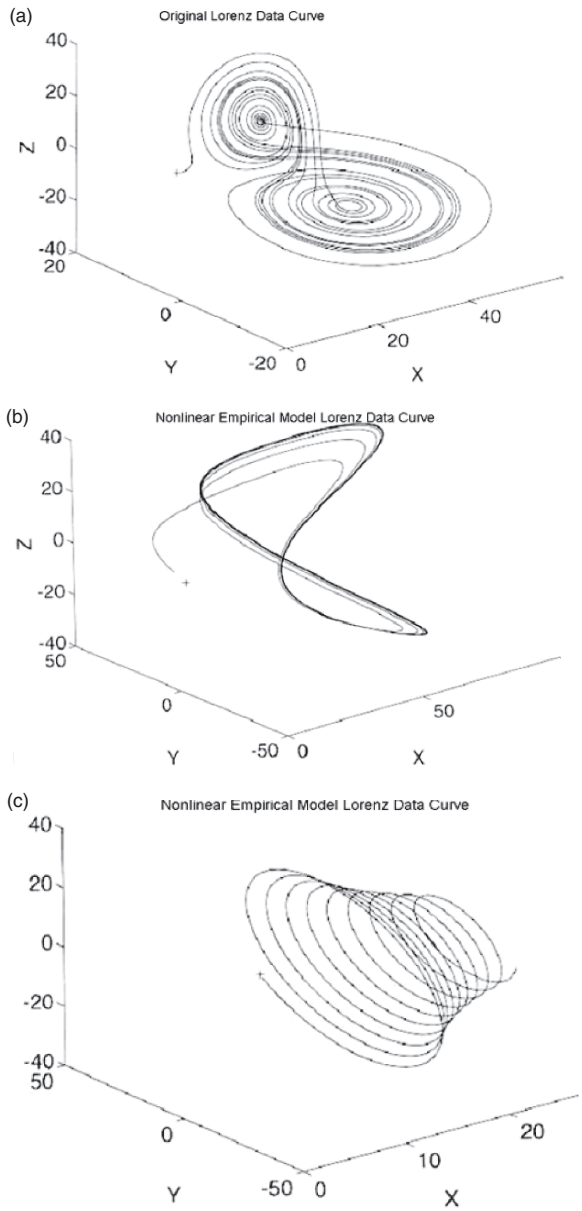


Fig. 18.2 The Lorenz butterfly attractor. (a) Initial data created from ( ) with Runge Kutta integration, (b) nonlinear model fit with a GA, and (c) linear model fit

This example has shown that nonlinear empirical models show promise for capturing the essential dynamics of nonlinear systems. Such a nonlinear empirical model, however, is not easily fit using analytical or traditional techniques. Instead, this model

relied on a GA to optimize the match with the model-produced data.

18.4 Example 2 – Least Squares Assimilation

18.4.1 Motivation

In this section, we wish to assimilate chemical, biological, radiological, or nuclear (CBRN) concentrations into a simple wind model that then forces a transport and dispersion model to forecast contaminant concentration. There is a long history of assimilating monitored data into meteorological models (Daley 1991; Kalnay 2003). In most cases, the goal is to assimilate data observations into the model fields so that the analyzed field is consistent with the model physics. Most methods work with observed quantities that are either the same fields as those being predicted or ones that can be readily transformed into the predicted quantities. For the CBRN problem, however, the observed quantity is concentration, but the wind field must be modeled to predict a concentration closer to that observed. Therefore, it requires inverting a full transport and dispersion model to relate wind field to concentration. Thus, using concentration data to assimilate the wind field is a complex problem. Chapter 14 showed how a GA could be used to back-calculate better constant wind direction and source parameters if field monitored concentrations are available. The goal here is to use concentration data to infer a time varying wind field and to use that field to predict the subsequent transport and dispersion.

Specifically, we seek to optimize the wind direction to produce a predicted concentration field closest to that monitored. This method is most akin to the variational approaches to assimilation, but rather than using the variational formalism, "leaps" directly to computing the best match. Again, we choose the continuous GA as our optimization tool. In this case, we assume that the source characteristics are known and seek to compute the time evolving wind direction. We set up an identical twin experiment in which a meandering plume is forced by a sinusoidally varying wind field. The domain is 5×5 km with sensors sited every 125 m. The synthetic data is produced with an integration time interval of 20 s. The surface level meandering plume is plotted on the fully resolved 40×40 grid in Fig. 18.3a. We will use these data to assess two approaches to assimilating the wind field: (1) static and (2) dynamic assimilation.

18.4.2 Genetic Algorithm Assimilation – Static Approach

The static approach uses the spatially varying wind field to produce a meandering concentration field, as would be the case where local terrain affects the flow.

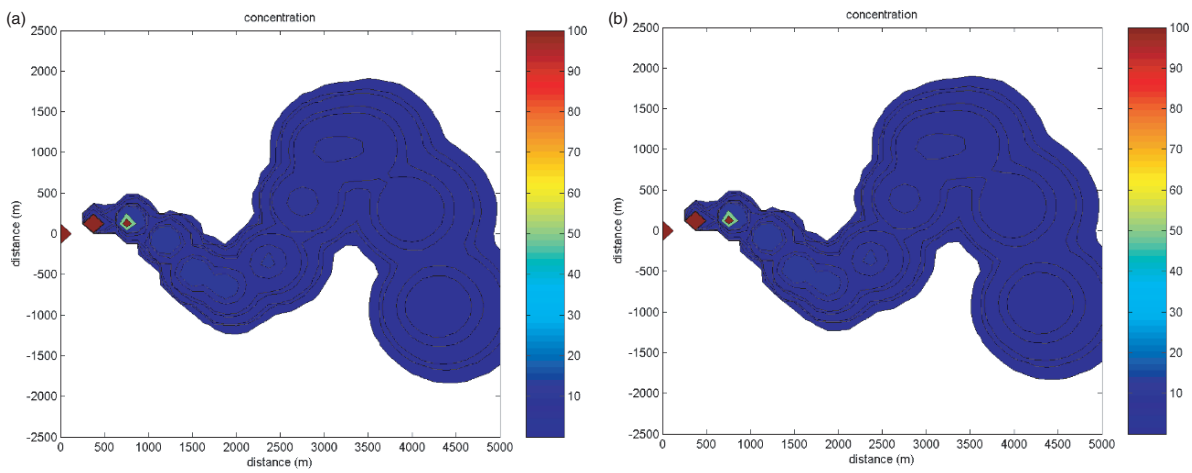


Fig. 18.3 Static assimilation of a meandering plume. (a) Truth and (b) GA computed plume

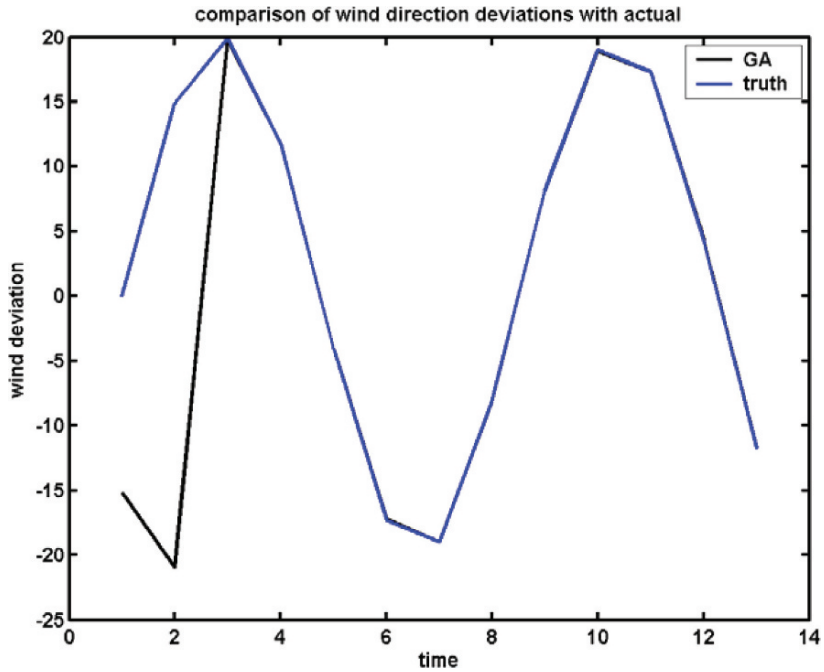


Fig. 18.4 GA match to the actual wind direction

In this case a fit to the Lagrangian evolution of the plume concentration field is accomplished in a single GA solution. We presume that the entire field is available *a priori* and seek to find the wind direction at each time. In this case, the wind speed is held constant. The first experiment used all of the sensor data and sampled the plume every four time steps (80 s). Figure 18.3b indicates the resulting plume. The plots are nearly indistinguishable from the sampled data in Fig. 18.3a. The GA match to the actual wind direction is indicated in Fig. 18.4. Except for the two initial times, the two are nearly identical. The lack of agreement at the initial time reflects the fact that the plume at that time had not yet produced any concentrations – i.e., the problem at that time is ill-posed. The second calculation overshoots to push the computed direction toward the exact solution. Figure 18.5 shows the GA convergence. The cost function continues to decrease, indicating that we could continue to iterate the GA to get increasingly more accurate solutions.

What if we do not have such a dense sensor network? Figure 18.6a shows the results of sampling one in four of the sensors, every 500 m. Although the plume is coarse, it accurately reproduces the location of the plume when compared to Fig. 18.3a. The very coarse network in Fig. 18.6b shows the limitations of the model when sampling every eight points (1,000 m). Even in this data starved situation, however, the basic shape of the plume was captured.

18.4.3 Genetic Algorithm Assimilation – Dynamic Approach

The second approach to assimilation is dynamic in nature. We assume that we have already modeled the past plume history and seek to assimilate the most recent concentration observations into the transport and dispersion model with the GA approach to optimizing the wind direction. This is a two step process: (1) we use the current concentration measurements to compute the optimal wind direction and (2) we use that wind direction to forecast the location of the plume at the next observation time. This dynamic process is closer to how such a process would progress in real time. The results at full resolution appear in Fig. 18.7, which should be compared Fig. 18.3a. Here we assume full spatial resolution but a temporal observation spacing of every four time periods (80 s). Although the shape of the plume is not bad, we note a phase shift in the plume meander by the distance corresponding

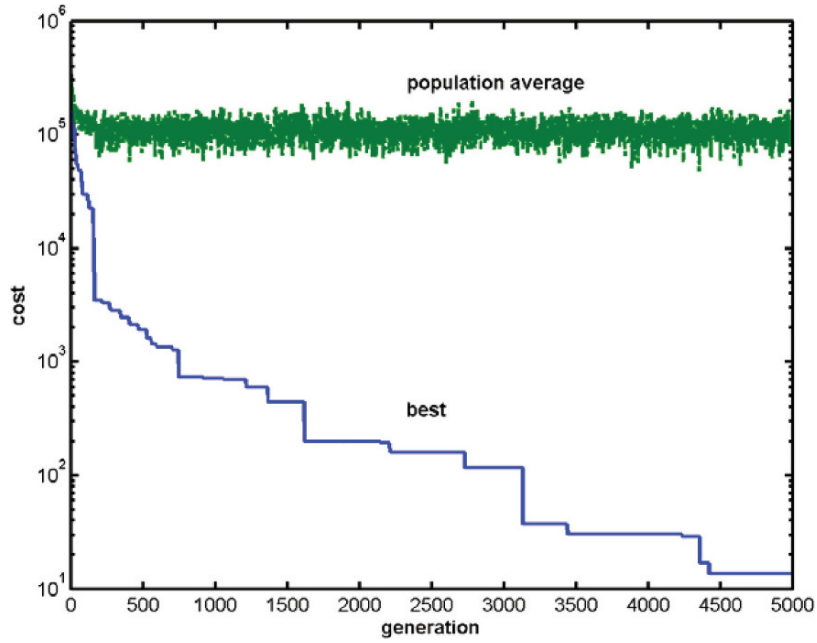


Fig. 18.5 Convergence of GA solution

to the sampling time interval. This phase shift is expected since the next time concentration can only reflect the most recent wind calculation. As the plume grows in time, the region that is impacted varies more. That problem could be addressed through drifting the observations with the wind; that is, presupposing the transport of the contaminant at the current wind speed. Figure 18.8 shows the degree of match between the dynamically computed wind direction and the true wind direction. Note the discrepancy in two aspects. At the beginning of the calculation, the match is poor. That discrepancy is explained by inadequate sampling of the plume before it is large enough to span multiple sensors. As the plume grows, the correspondence with the true wind direction improves. At the final sampling times, however, the difference between computed wind direction and truth degrades. This degradation occurs because of the inherent phase shift in the assimilation of spatially fixed observations. In a real situation, such rapid wind shifts may be less likely.

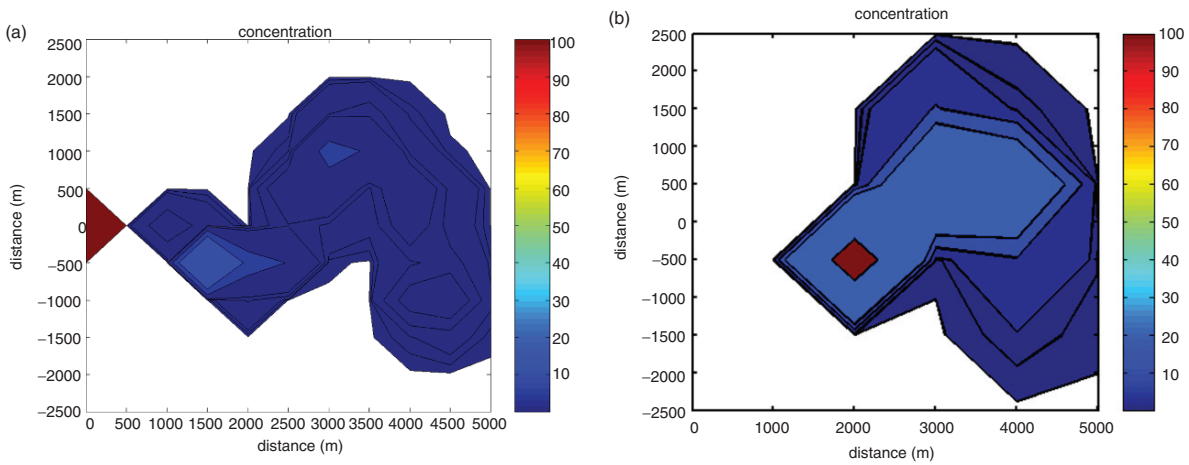


Fig. 18.6 GA computed plume when the spatial sampling is (a) 1:4 and (b) 1:8

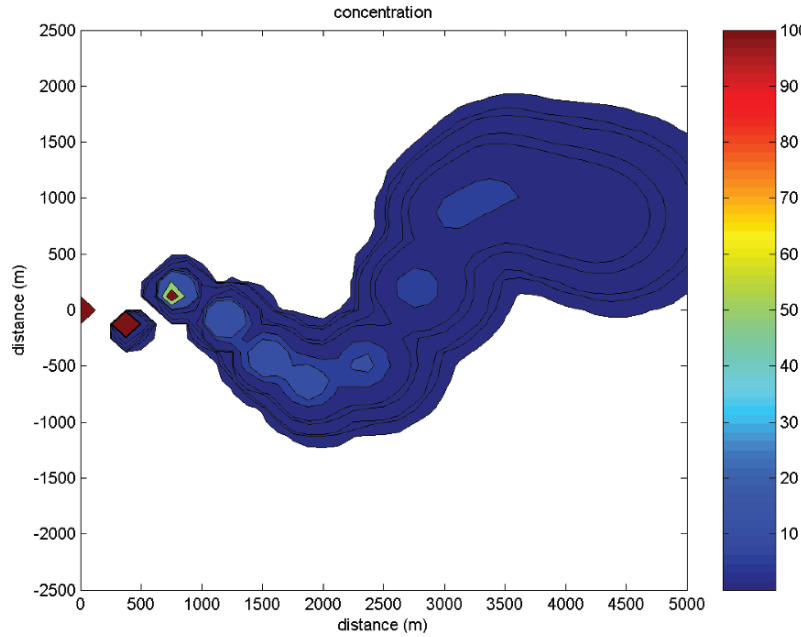


Fig. 18.7 Dynamic assimilation of the meandering plume

This assimilation example is indicative of the ways in which AI could help solve problems that are difficult for standard techniques. In this case, using concentration data to assimilate wind field involves a transport and dispersion model that is too complex for some standard methods such as variational approaches. The GA is successful at simplifying the problem by "leaping" over the formalism and finding an operationally useful solution.

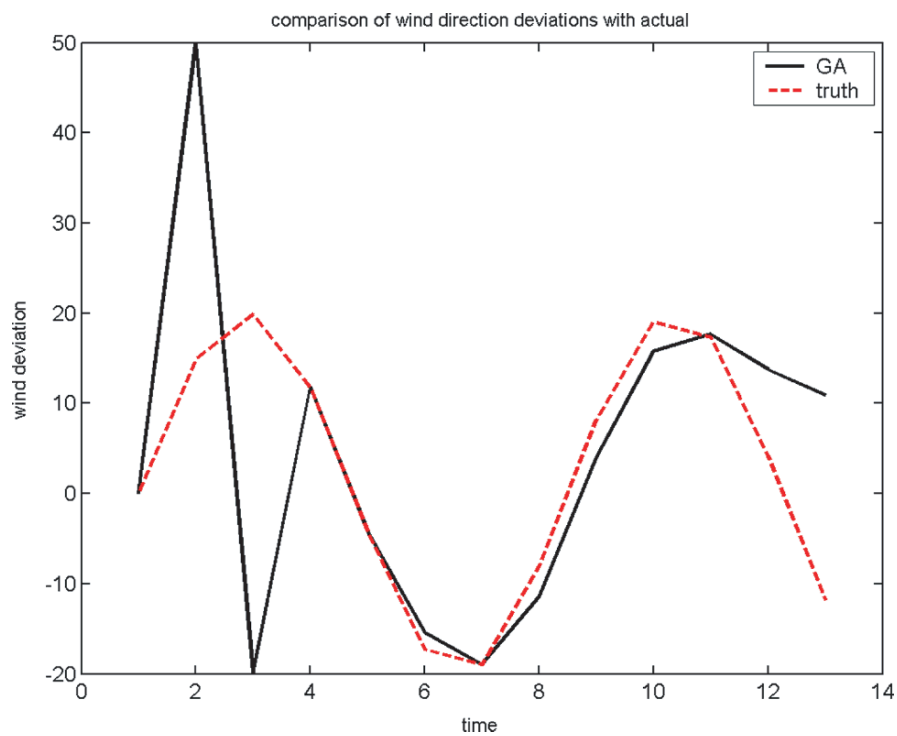


Fig. 18.8 Comparison of dynamically computed wind direction with the actual wind direction

18.5 Example 3 – Using GAs and NNs for Classification

18.5.1 Introduction

It is common in environmental prediction to separate events into discrete classes, such as whether it will hail or not or whether precipitation will be rain or snow. Generally, many such classification models are based on the minimization of cross-entropy (Bishop 1996), which is continuous and differentiable in model parameters. Most common measures of classification performance, however, are based on the contingency table (see Chapter 3). The critical success index is one example that is common in meteorology; numerous others are discussed in Chapter 3 and by Marzban (1998). Because, they are based on a discrete table – the contingency table – they are neither continuous nor differentiable in the model parameters. This is why they are referred to as discrete measures. Given that the final model is to be assessed in terms of such discrete measures, it is natural to optimize them directly.

In this section, we compare a number of such measures that are optimized directly using GAs: the results are compared to the more traditional approach of first minimizing cross entropy, followed by the optimization of a discrete performance measure. One may interpret this task as equivalent to the comparison of two parameter estimation (or training) algorithms – one capable of optimizing discrete measures, the other not. It should be noted that, strictly speaking, the latter is not an optimization algorithm because it is a two-stage procedure with different measures optimized at each stage.

18.5.2 Methodology

The data set used for our comparison consists of over 21,000 cases with every case including three predictors and one predictand; the binary number labeling the occurrence or nonoccurrence of hail. The predictors are related to both Doppler radar-derived parameters and parameters representing the near-storm environment. This data set has been used in the development of neural networks to aid the National Severe Storms' Hail Detection Algorithm in detecting hail and estimating hail size (Marzban and Witt 2000, 2001). The neural network for predicting the occurrence of hail was trained by minimizing cross-entropy, and that for predicting the size of hail was based on the minimization of Mean Squared Error (MSE). Both of these measures are continuous and differentiable. Given that the latter network is a classifier, its performance was assessed in terms of discrete measures – specifically, the Critical Success Index (CSI) and the Heidke Skill Statistic (HSS). As such, its development was a two-stage process involving the maximization of cross-entropy followed by the maximization of the discrete measures. The discrete measures were computed from the contingency table, which in turn was formed by placing a threshold on the probability (of hail) produced by the network. As such, the maximization of the discrete measure (at the second stage) is tantamount to identifying the threshold which yields that highest performance. In addition to CSI and HSS, one other measure will be employed here – the fraction correct (FRC). This study is inherently empirical in that the findings are specific to the data set examined.

The parametric form of the model examined here is motivated by the neural network (see Chapter 2). Specifically, it is

$$y = g\left(\sum_{i=1}^{H} \omega_i f\left(\sum_{j=1}^{N_{in}} \omega_{ij} x_j - \theta_j\right) - \omega\right) + \varepsilon \tag{18.7}$$

where the ω's and θ's are all parameters to be estimated from data. For clarity, the index $i = 1, 2, \ldots, N$, referring to the data case is not shown. N_{in} refers to the number of predictors, and H is a parameter that gauges the nonlinearity of the function. Note that H is analogous to the order of a polynomial. Although there are techniques for estimating it from data, the optimal value for H is not important in the current study. It will be fixed at $H = 2$, and 4. Recall that the goal of the study is to examine different training algorithms, rather than to develop the "best" model for hail detection.

A neural network is trained by two different training algorithms – conjugate gradient and genetic algorithm.

(a) Minimize cross-entropy using a conjugate gradient method (Press et al. 1999) to build a model producing a continuous predictand (i.e. probability).

Place a threshold on the predictand in order to construct a contingency table, and compute performance measures. Vary the threshold across the full range of the predictand in order to identify the threshold at which the maximum performance measure occurs. This maximum value is taken to represent optimal performance in this approach. This method minimizes cross-entropy, which means that the network produces a continuous quantity between 0 and 1. This output is then dichotomized by the introduction of a threshold, and the discrete measure is optimized as a function of this threshold.

(b) Maximize the discrete measure, directly, by using a genetic algorithm. Chapter 5 introduces genetic algorithms and their advantages. The primary advantage of importance to the current application is its ability to directly optimize non-continuous, non-differentiable measures. The GA used for this study is a continuous parameter GA. This training algorithm maximizes the discrete measure directly. It also directly optimizes the threshold as part of the calculation.

One common feature of both conjugate gradient and genetic algorithm is that they are iterative. One typically begins with a random set of values for the parameters, and applies the training algorithm until the performance measure converges to some optimal value. To assure that the training algorithms are not trapped in shallow local minima, both CG and GAs are applied to five different initial parameter values.

The performance measures are detailed in Chapter 3 of this volume: FRC, CSI, and HSS.

18.5.3 Results

Figure 18.9a through 18.9f display the results. The horizontal lines correspond to the performance measures reached by the GA for five different initializations. They are not a function of threshold since the threshold is optimized directly by this approach. The remaining five curves correspond to the performance measures based on the minimization of cross-entropy by CG. It can be seen that the curves for all three measures display similar behavior, although the CG curves reach their peaks at somewhat different values of the threshold. The behavior of these curves is in complete agreement with their theoretical behavior in Gaussian models (Marzban 1998).

Table 18.1 The average performance values and confidence intervals, for the different measures, and for $H = 2$ and $H = 4$

Measure	GA	CG
	$H = 2$	
FRC	0.92163 ± 0.00019	0.92130 ± 0.00006
CSI	0.50370 ± 0.00105	0.49970 ± 0.00037
HSS	0.62076 ± 0.00266	0.61768 ± 0.00038
	$H = 4$	
FRC	0.92157 ± 0.00022	0.92109 ± 0.00018
CSI	0.50360 ± 0.00147	0.50208 ± 0.00146
HSS	0.62192 ± 0.00158	0.61958 ± 0.00106

The important point of these figures is that the traditional two-stage optimization of the measures does not yield performance values as high as those obtained by their direct optimization via the GA. This is true of all five curves. Thus, the GA has an advantage over the alternatives that require continuous and differentiable performance measures.

The difference between the two approaches, however, is rather small in that the curves approach and even cross some of the horizontal lines. The question arises as to whether the difference is statistically significant. To that end, a t-test is performed and 2σ confidence intervals are computed. Note that 2σ corresponds to a 97% confidence interval, displayed in Table 18.1. The t-values (not shown in table) for the $H = 2$ case are in the 2.8 range, and those of the $H = 4$ case are in the 2.4 range. It can be seen that the differences between the two approaches are statistically significant. The slightly lower t-value of the $H = 4$ case is anticipated from the larger variations between the five curves in the right figures in Fig. 18.9.

This study has shown that when a neural network is trained directly using the performance measure that will be used to judge its success, it is somewhat more skillful than if trained using the traditional mean square error approach. To train the network using the discrete performance measures, however, requires use of an artificial intelligence technique that can work with discrete numbers. The GA produced better performance measures than the traditional conjugate gradient approach of optimizing cross-entropy because it directly optimizes those measures on which it is

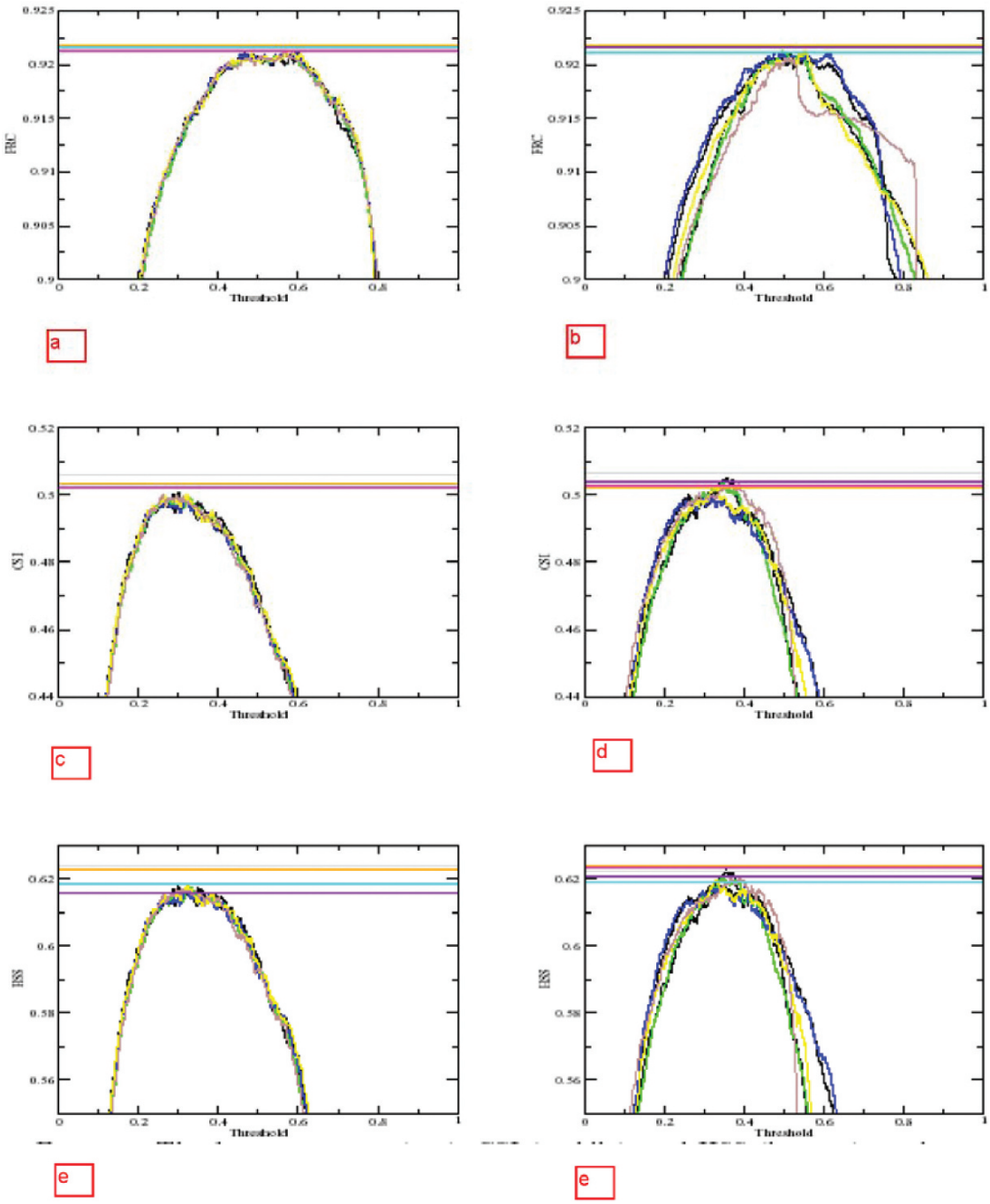


Fig. 18.9 The fraction correct (top), CSI (middle), and HSS (bottom) as obtained from five different initializations of conjugate gradient with $H = 2$ (left) and $H = 4$ (right). The horizontal lines are the corresponding scores from GA that includes optimizing the threshold

being judged. We should note that the genetic algorithm does take considerably more CPU time to complete the optimization of the neural network weights than the competing methodology. It can, however, optimize the cutoff threshold at the same time, eliminating the need for testing on various thresholds.

The results reported here used a single data set; therefore, they should be considered preliminary until they can be confirmed with other data sets of differing types and sizes. We do expect that with more experiments we will be able to generalize these results to other cases.

18.6 Example 4 – Solving PDEs

As a final example, we show how a GA can be used to solve a high order nonlinear partial differential equation that is formally nonintegrable. Normally, we don't think of ordinary and partial differential equations (ODEs and PDEs) as minimization problems. Thus, this is an example of how common difficult problems can be posed as one in optimization. Koza (1992) demonstrated that a GA could solve a simple differential equation by minimizing the value of the solution at 200 points. To do this, he numerically differentiated at each point and fit the appropriate solution using a GA. Karr et al. (2001) used GAs to solve inverse initial boundary value problems and found a large improvement in matching measured values. That technique was demonstrated on elliptic, parabolic, and hyperbolic PDEs.

We examine the Super Korteweg-de Vries equation (SKDV), a fifth-order nonlinear partial differential equation. It has been used as a model for shallow water waves near a critical value of surface tension, among other phenomena (Yoshimura and Watanabe 1982). This equation admits solitary waves and their periodic generalizations, cnoidal waves. Solitary waves, or solitons, are permanent-form waves for which the nonlinearity balances the dispersion to produce a coherent structure. The SKDV can be written as:

$$u_t + \alpha u u_x + \mu u_{xxx} - \nu u_{xxxxx} = 0 \tag{18.8}$$

The functional form of the wave is denoted by u; time derivative by the t subscript; spatial derivative by the x subscript; and α, μ, and ν are parameters of the problem taken as $\nu = 1$, $\mu = 0$, $\alpha = 1$. We wish to solve for waves that are steadily translating, so we write the t variation using a Galilean transformation, $X = x - ct$, where c is the phase speed of the wave. The phase speed and amplitude of solitary-type waves are interdependent. Here, we use a phase speed of $c = 14.683$ to match with a well known approximate nonlinear solution (Boyd 1986). Thus, our SKDV becomes a fifth-order, nonlinear ODE:

$$(\alpha u - c)u_X + \mu u_{XXX} - \nu u_{XXXXX} = 0 \tag{18.9}$$

Boyd (1986) extensively studied methods of solving this equation. He expanded the solution in terms of Fourier series to find periodic cnoidal wave solutions (solitons that are repeated periodically). Among the methods used are the analytical Stokes' expansion, which intrinsically assumes small amplitude waves, and the numerical Newton-Kantorovich iterative method, which can go beyond the small amplitude regime if care is taken to provide a very good first guess. Haupt and Boyd (1988) extended these methods to deal with resonance conditions. These methods, however, require careful analytics and programming that is very problem specific. Here, we instead cast the problem as one in optimization and solve with a genetic algorithm.

To find the solution of equation (18.9), we expand the function u in terms of a Fourier cosine series to K terms to obtain the approximation, u_K:

$$u(X) \simeq u_K(X) = \sum_{k=1}^{K} a_k \cos(kX) \tag{18.10}$$

The cosine series assumes that the function is symmetric about the X-axis (without loss of generality). In addition, we use the "cnoidal convention" by assuming that the constant term, a_0 is 0. Now, we can easily take derivatives as powers of the wave numbers to write the cost function that we wish to minimize as:

$$\text{cost}(u_K) = \sum_{k=1}^{K} \left[-k(\alpha u - c) + k^3\mu + k^5\nu\right] a_k \sin(kx) \tag{18.11}$$

We wish to find the coefficients of the series, a_k. Note that we must compute u to insert into the cost function (18.11).

We computed the coefficients, a_k, to find the best cnoidal wave solution for $K = 6$. We used $N_{pop} = 100$, $\mu = 0.2$, and 70 iterations. We evaluated the cost function at grid points and summed their absolute value. The results appear in Fig. 18.10. The solid line is the "exact" solution reported by Boyd (1986)

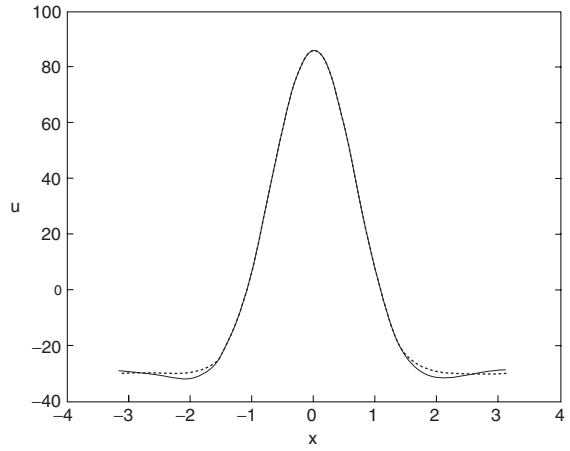


Fig. 18.10 Double cnoidal wave of the SKDV equation. Solid: perturbation solution, dashed: GA solution

and the dashed line is the GA's approximation to it. They are barely distinguishable. In addition, we show a GA solution that converged to a double cnoidal wave as Fig. 18.11. Such double cnoidal waves are very difficult to compute using other methods (Haupt and Boyd 1988).

So we see that GAs show promise for finding solutions of differential and partial differential equations, even when these equations are highly nonlinear and have high-order derivatives, thus being difficult to solve with more traditional techniques.

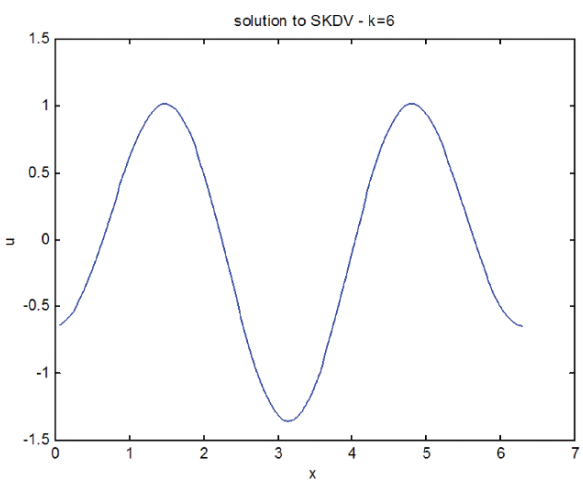


Fig. 18.11 Double cnoidal wave of the SKDV equation as found by the GA

18.7 Summary and Conclusions

Genetic algorithms have begun to find their way into applications in the many disciplines of environmental science, but their strengths have only begun to be tapped. We have shown here how versatile these algorithms are at finding solutions, even where other methods often fail.

Several specific applications have been described, including building empirical models of data, using a genetic algorithm to train a Neural Network for a classification problem, data assimilation with a genetic algorithm, and even using a genetic algorithm to find steadily translating solutions of a highly nonlinear fifth order PDE. Some of the references of Section 18.2 have used other types of GAs in highly imaginative ways and combined them with other techniques. The author and her colleagues are involved in other new application areas as well. One such application uses a genetic algorithm to partition clusters of weather forecasts to optimize which linear combination of ensemble members is likely to make the best prediction for a particular regime.

The use of GAs is growing. In some fields, such as groundwater work, they are becoming a common tool. For other areas, they have not yet been considered. For instance, the simple example of Section 18.3.3 might be a precursor of fitting parameters of full climate models. Linear fits give a good first fit to climate model data. Branstator and Haupt (1998) demonstrated that a simple linear stochastic model can reproduce responses to forcing better than a traditional linearized quasigeostrophic model that includes specific dynamics. But the linear techniques fall apart on highly nonlinear problems. That is where the genetic algorithm can be useful. If we can formulate a nonlinear form for a climate model in ways parallel to the Section 18.3 problem, we can use the GA to estimate the coefficients. Even the highly nonlinear Lorenz equations in the strange attractor regime can be fit to a quadratically nonlinear model using this technique (Haupt 2006). One can imagine that as we attempt to model larger nonlinear systems, techniques from artificial intelligence, such as genetic algorithms, could become useful to determining stochastic fits where the dynamical theory becomes intractable.

The environmental sciences have advanced through observing data and formulating models. Many of those models have used first physics principals. But many such models have reached a level of complexity that is

overwhelming. Now is when we return to finding other methods to interpret data. AI can aid in this quest. GAs are one such AI technique that can help develop new models of physical processes gleaned from observed data.

Acknowledgements The author wishes to thank Randy L. Haupt, who coauthored the genetic algorithm used for most of these applications. The nonlinear empirical model work was inspired by joint work with Jeffrey Weiss. Assimilation models resulted from work being done jointly with George S. Young, Anke Beyer, and Kerrie J. Long. Caren Marzban is coauthor of the work reported in Section 18.4.

References

Aly, A. H., & Peralta, R. C. (1999a). Comparison of a genetic algorithm and mathematical programming to the design of groundwater cleanup systems. *Water Resources Research*, *35*(8), 2415–2425.

Aly, A. H., & Peralta, R. C. (1999b). Optimal design of aquifer clean up systems under uncertainty using a neural network and a genetic algorithm. *Water Resources Research*, *35*(8), 2523–2532.

Babb, D. M., Verlinde, J., & Albrecht, B. A. (1999). Retrieval of cloud microphysical parameters from 94-GHz radar Doppler power spectra. *Journal of Atmospheric and Oceanic Technology*, *16*, 489–503.

Barth, H. (1992). Oceanographic experiment design II: Genetic algorithms. *Journal of Oceanic and Atmospheric Technology*, *9*, 434–443.

Bishop, C. M. (1996). *Neural networks for pattern recognition* (pp. 482). Oxford: Clarendon Press.

Boschetti, F., Dentith, M. C., & List, R. D. (1995). A staged genetic algorithm for tomographic inversion of seismic refraction data. *Exploration Geophysics*, *26*, 331–335.

Boschetti, F., Dentith, M. C., & List, R. D. (1996). Inversion of seismic refraction data using genetic algorithms. *Geophysics*, *61*, 1715–1727.

Boschetti, F., Dentith, M. C., & List, R. (1997). Inversion of potential field data by genetic algorithms. *Geophysical Prospecting*, *45*, 461–478.

Boyd, J. B. (1986). Solitons from sine waves: Analytical and numerical methods for nonintegrable solitary and cnoidal waves. *Physica*, *21D*, 227–246.

Branstator, G., & Haupt, S. E. (1998). An empirical model of barotropic atmospheric dynamics and its response to tropical forcing. *Journal of Climate*, *11*, 2645–2667.

Chan Hilton, A. B., & Culver, T. B. (2000). Constraint handling for genetic algorithms in optimal remediation design. *Journal of Water Resources Planning and Management*, *126*(3), 128–137.

Cartwright, H. M., & Harris, S. P. (1993). Analysis of the distribution of airborne pollution using gas. *Atmospheric Environment*, *27A*, 1783–1791.

Charbonneau, P. (1995). Genetic algorithms in astronomy and astrophysics. *The Astrophysical Journal Supplement Series*, *101*, 309–334.

Chunduru, R. K., Sen, M. K., Stoffa, P. L., & Nagendra, R. (1995). Non-linear inversion of resistivity profiling data for some regular geometrical bodies. *Geophysical Prospecting*, *43*, 979–1003.

Chunduru, R. K., Sen, M. K., & Stoffa, P. L. (1997). Hybrid optimization for geophysical inversion. *Geophysics*, *62*(4), 1196–1207.

Daley, R. (1991). *Atmospheric data assimilation* (457 pp.). Cambridge: Cambridge University Press.

Fang, H., Liang, S., & Kuusk, A. (2003). Retrieving leaf area index using a genetic algorithm. *Remote Sensing of Environment*, *85*, 257–270.

Fayad, H. (2001). *Application of neural networks and genetic algorithms for solving conjunctive water use problems*. Ph.D. dissertation, Utah State University, Logan, UT, 152 pp.

Garcia-Orellan, C. J., Macias-Macias, M., Serrano-Perez, A., Gonzalez-Velasco, H. M., & Gallardo-Caballero, R. (2002). A comparison of PCA, ICA and GA selected features for cloud field classification. *Journal of Intelligent and Fuzzy Systems*, *12*, 213–219.

Gonzalez, A., Perez, J. C., Herrera, F., Rosa, F., Wetzel, M. A., Borys, R. D., et al. (2002). Stratocumulus properties retrieval method from NOAA-AVHRR data based on the discretization of cloud parameters. *International Journal of Remote Sensing*, *23*, 627–645.

Hart, J., Hunt, I., & Shankararaman, V. (1998). Environmental management systems – A role for AI? in *Workshop Binding Environmental Sciences and Artificial Intelligence (BESAI'98)*, edited by U. Cortés and M. Sànchez-Marrè, pp. 1–10.

Hassan, R. A., & Crossley, W. A. (2003). Multi-objective optimization for communication satellites with a two-branch tournament genetic algorithm. *Journal of Spacecraft Rockets*, *40*(2), 266–272.

Hasselman, K. (1988). Pips and pops: The reduction of complex dynamical systems using principal interaction and oscillation patterns. *Journal of Geophysical Research*, *93*, 11015–11021.

Haupt, S. E. (2006). Nonlinear empirical models of dynamical systems. *Computers and Mathematics with Applications*, *51*, 431–440.

Haupt, S. E., & Boyd, J. P. (1988). Modeling nonlinear resonance: A modification to the stokes' perturbation expansion. *Wave Motion*, *10*, 83–98.

Jervis, M., Sen, M. K., & Stoffa, P. L. (1996). Prestack migration velocity estimation using nonlinear methods. *Geophysics*, *60*, 138–150.

Kalnay, E. (2003). *Atmospheric modeling, data assimilation and predictability* (pp. 136–204). Cambridge: Cambridge University Press.

Karr, C. L., Yakushin, I., & Nicolosi, K. (2001). Solving inverse initial-value, boundary value problem via GA. *Engineering Applications of Artificial Intelligence*, *13*, 625–633.

Kim, S., Lee, H., Kim, J., Kim, C., Ko, J., Woo, H., et al. (2002). Genetic algorithms for the application of Activated Sludge Model No. 1. *Water Science and Technology*, *45*(4–5), 405–411.

Kishtawal, C. M., Basu, S., Patadia, F., & Thapliyal, P. K. (2003). Forecasting summer rainfall over India using genetic algorithm. *Geophysical Research Letters*, *30*, 2203. Doi:10.1029/2003GL018504.

Kondrashov, D., Kravtsov, S., Robertson, A. W., & Ghil, M. (2005). A hierarchy of data-based ENSO models. *Journal of Climate*, *18*, 4404–4424.

Koza, J. R. (1992). Chapter 10: The genetic programming paradigm: Genetically breeding populations of computer programs to solve problems. In B. Soucek (Ed.), *Dynamic, genetic, and chaotic programming* (pp. 203–321), The Sixth Generation. New York: Wiley.

Lakshmanan, V. (2000). Using a genetic algorithm to tune a bounded weak echo region detection algorithm. *Journal of Applied Meteorology*, *39*, 222–230.

Loughlin, D. H., Ranjithan, S. R., Baugh, J. W., Jr., & Brill, E. D., Jr. (2000). Application of genetic algorithms for the design of ozone control strategies. *Journal of the Air & Waste Management Association*, *50*, 1050–1063.

Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, *20*, 130–141.

McCullagh, J., Choi, B., & Bluff, K. (1999). Genetic synthesis of a neural network for rainfall and temperature estimations. *Australian Journal for Intelligent Information Processing System*, *5*(3), 187–193.

McKinney, D. C., & Lin, M. D. (1993). Genetic algorithm solution of ground water management models. *Water Resources Research*, *30*(6), 3775–3789.

Marzban, C., & Haupt, S. E. (2005). On genetic algorithms and discrete performance measures. *AMS 4th Conference on Artificial Intelligence*, San Diego, CA, paper 1.1.

Marzban, C., & Witt, A. (2000). *Bayesian neural networks for severe hail prediction. A report*. Available at http://www.nhn.ou.edu/\∼{}marzban/hda_class.pdf.

Marzban, C., & Witt, A. (2001). A Bayesian neural network for hail size prediction. *Weather Forecasting*, *16*(5), 600–610.

Marzban, C. (1998). Bayesian probability and scalar performance measures in Gaussian models. *Journal of Applied Meteorology*, *37*, 72–82.

Menke, W. (1984). *Geophysical data analysis: Discrete inverse theory*. New York: Academic Press.

Minister, J. B. H., Williams, N. P., Masters, T. G., Gilbert, J. F., & Haase, J. S. (1995). Application of evolutionary programming to earthquake hypocenter determination. In *Evolutionary programming: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pp. 3–17.

Mohan, S., & Loucks, D. P. (1995). Genetic algorithms for estimating model parameters. *Integrated Water Resources Planning for the 21st century, Proceedings of the 22nd Annual Conference*, ASCE, Cambridge, MA.

Mulligan, A. E., & Brown, L. C. (1998). Genetic algorithms for calibrating water quality models. *Journal of Environmental Engineering*, 202–211.

Penland, C. (1989). Random forcing and forecasting using principal oscillation pattern analysis. *Monthly Weather Review*, *117*, 2165–2185.

Penland, C., & Ghil, M. (1993). Forecasting northern hemisphere 700 mb geopotential height anomalies using empirical normal modes. *Monthly Weather Review*, *121*, 2355–2372.

Penland, C., & Magorian, T. (1993). Prediction of NINO3 sea-surface temperatures using linear inverse modeling. *Journal of Climate*, *8*, 1067–1076.

Penland, C., & Matrosova, L. (1994). A balance condition for stochastic numerical models with application to the El Nino-Southern Oscillation. *Journal of Climate*, *7*, 1352–1372.

Popov, M., He, S., & Thottappillil, R. (2000). Reconstruction of lightning currents and return stroke model parameters using remote electromagnetic fields. *Journal of Geophysical Research*, *105*, 24469–24481.

Porsani, M. J., Stoffa, P. L., Sen, M. K., & Chunduru, R. K. (2000). Fitness functions, genetic algorithms and hybrid optimization in seismic waveform inversion. *Journal of Seismic Exploration*, *9*, 143–164.

Porto, V. W., Fogel, D. B., & Fogel, L. J. (1995). Alternative neural network training methods. *IEEE Expert System*, June, *10*(3), 16–22.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1999). *Numerical recipes in C* (pp. 994). Cambridge: Cambridge University Press.

Recknagel, F. (2001). Applications of machine learning to ecological modelling. *Ecological Modelling*, *146*, 303–310.

Ritzel, B. J., Eheart, J. W., & Rajithan, S. (1994). Using genetic algorithms to solve a multiple objective groundwater pollution containment problem. *Water Resources Research*, *30*(5), 1589–1603.

Rogers, L. L., & Dowla, F. U. (1994). Optimization of groundwater remediation using artificial neural networks with parallel solute transport modeling. *Water Resources Research*, *30*(2), 457–481.

Sen, M. K., & Stoffa, P. L. (1992a). Rapid sampling of model space using genetic algorithms: Examples from seismic waveform inversion. *Geophysical Journal International*, *108*, 281–292.

Sen, M. K., & Stoffa, P. L. (1992b). Genetic inversion of AVO. *Geophysics: The Leading Edge of Exploration*, *11*(1), 27–29.

Sen, M. K., & Stoffa, P. L. (1996). Bayseian inference, gibbs' sampler and uncertainty estimation in geophysical inversion. *Geophysical Prospecting*, *44*, 313–350.

Sen, Z., & Oztopal, A. (2001). Genetic algorithms for the classification and prediction of precipitation occurrence. *Hydrological Sciences*, *46*(2), 255–268.

Shieh, H.-J., & Peralta, R. C. (1997). Optimal system design of in-situ bioremediation using genetic annealing algorithm. In *Ground Water: An Endangered Resource, Proceedings of Theme C, Water for a Changing Global Community, 27th Annual Congress of the International Association of Hydrologic Research*, pp. 95–100.

Simpson, A. R., Dandy, G. C., & Murphy, L. J. (1994). Genetic algorithms compared to other techniques for pipe optimization. *Journal of Water Resources Planning and Management*, *120*(4), 423–443.

Vrugt, J. A., Gupta, H. V., Bastidas, L. A., Bouten, W., & Sorooshian, S. (2003). Effective and efficient algorithm for multiobjective optimization of hydrologic models. *Water Resources Research*, *39*(8), 1214. Doi:10.1029/2002WR001746.

Yashimura, K., & Watanabe, S. (1982). Chaotic behavior of nonlinear evolution equation with fifth order dispersion. *Journal of the Physical Society of Japan*, *51*, 3028–3035.

Yin, C., Liu, X., Guo, W., Lin, T., Wang, X., & Wang, L. (2002). Prediction and application in QSPR of aqueous solubility of sulfur-containing aromatic esters using GA-based MLR with quantum descriptors. *Water Research*, *36*, 2975–2982.

Machine Learning Applications in Habitat Suitability Modeling

19

Sašo Džeroski

19.1 Introduction

Environmental sciences comprise the scientific disciplines, or parts of them, that consider the physical, chemical and biological aspects of the environment (Allaby 1996). Environmental sciences are possibly the largest grouping of sciences, drawing heavily on life sciences and earth sciences, both of which are relatively large groupings themselves. Life sciences deal with living organisms and include (among others) agriculture, biology, biophysics, biochemistry, cell biology, genetics, medicine, taxonomy and zoology. Earth sciences deal with the physical and chemical aspects of the solid Earth, its waters and the air that envelops it. Included are the geologic, hydrologic, and atmospheric sciences. The latter are concerned with the structure and dynamics of Earth's atmosphere and include meteorology and climatology.

The field of environmental science is very interdisciplinary. It exists most obviously as a body of knowledge on its own right when a team of specialists assembles to address a particular issue (Allaby 1996). For instance, a comprehensive study of a particular stretch of a river would involve determining the geological composition of the riverbed (geology), determining the chemical and physical properties of the water (chemistry, physics), as well as sampling and recording the species living in and near the water (biology). Environmental sciences are highly relevant to environmental management, which is concerned with directing human activities that affect the environment.

Sašo Džeroski (✉)
Jozef Stefan Institute, Department of Knowledge Technologies, Jamova 39, 1000 Ljubljana, Slovenia
email: Saso.Dzeroski@ijs.si

The most typical representative of environmental sciences is ecology, which studies the relationships among members of living communities and between those communities and their abiotic (non-living) environment. Ecology is frequently defined as the study of the distribution and abundance of plants and animals (e.g., Krebs 1972). The distribution can be considered along the spatial dimension(s) and/or the temporal dimension.

Within ecology, the topic of ecological modeling (Joergensen and Bendoricchio 2001) is rapidly gaining importance and attention. Ecological modeling is concerned with the development of models of the relationships among members of living communities and between those communities and their abiotic environment. These models can then be used to better understand the domain at hand or to predict the behavior of the studied communities and thus support decision making for environmental management. Typical modeling topics are population dynamics of several interacting species and habitat suitability for a given species (or higher taxonomic unit).

Machine learning is one of the essential and most active research areas in the field of artificial intelligence. In short, it studies computer programs that automatically improve with experience (Mitchell 1997). The most researched type of machine learning is inductive machine learning, where the experience is given in the form of learning examples. Supervised inductive machine learning, sometimes also called predictive modeling, assumes that each learning example includes some target property, and the goal is to learn a model that accurately predicts this property.

Machine learning (and in particular predictive modeling) is increasingly often used to automate the construction of ecological models (Džeroski 2001). Most

S. E. Haupt et al. (eds.), *Artificial Intelligence Methods in the Environmental Sciences*

frequently, models of habitat suitability and population dynamics are constructed from measured data by using machine learning techniques. The most popular machine learning techniques used for modeling habitat suitability include decision tree induction (Breiman et al. 1984, see also Chapter 4 of this volume by Dattatreya), rule induction (Clark and Boswell 1991), and neural networks (Lek and Guegan 1999, see also Chapter 2 of this volume by Marzban).

In this chapter, we will focus on applications of machine learning in ecological modeling, more specifically, applications in habitat suitability modeling. Habitat-suitability modeling studies the effect of the abiotic characteristics of the habitat on the presence, abundance or diversity of a given taxonomic group of organisms. For example, one might study the influence of soil characteristics, such as soil temperature, water content, and proportion of mineral soil on the abundance and species richness of springtails, the most abundant insects in soil. To build habitat-suitability models, machine learning techniques can be applied to measured data on the characteristics of the environment and the abundance of the taxonomic group(s) studied.

In the remainder of this chapter, we first discuss in more detail the task of habitat suitability modeling. We next briefly describe two approaches to machine learning that are often used in habitat suitability modeling: decision tree induction and rule induction. We then give examples of using machine learning to construct models of habitat suitability for several kinds of organisms. These include habitat models for bioindicator organisms in a river environment, springtails and other soil organisms in an agricultural setting, brown bears in a forest environment, and finally habitat suitability models for sea cucumbers in a sustainable fishing setting.

19.2 Habitat Suitability Modeling

If ecology is defined as the study of the distribution and abundance of plants and animals, habitat suitability modeling is concerned with the spatial aspects of the distribution and abundance. Habitat suitability models relate the spatially varying characteristics of the environment to the presence, abundance or diversity of a given (taxonomic) group of organisms. For example, one might study the influence of soil characteristics, such as soil temperature, water content, and proportion of mineral soil on the abundance and species richness of springtails, the most abundant insects in soil.

The input to a habitat model is thus a set of environmental characteristics for a given spatial unit of analysis. The output is a target property of the given (taxonomic) group of organisms. Note that the size of the spatial unit, as well as the type of environmental variables, can vary considerably, depending on the context, and so can the target property of the population (even though to a lesser extent).

The spatial unit considered may be of different size for different habitat models. For example, in the study of Collembola habitat, the soil samples taken were of size 7.8 cm diameter and 5 cm depth (Kampichler et al. 2000), in the study of sea cucumber habitat transects of 2 by 50 m of the sea bed were considered (Džeroski and Drumm 2003), and in ongoing studies of potential habitats for different tree species under varying climate change scenarios, 1 by 1 km squares are considered (Ogris and Jurc 2007). Habitat models can thus operate at very different spatial scales.

The input to a habitat model is a set of environmental variables, which may be of three different kinds. The first kind concerns abiotic properties of the environment, e.g., physical and chemical characteristic thereof. The second kind concerns some biological aspects of the environment, which may be considered as en external impact on the group of organisms under study. Finally, the third kind of variables are related to human activities and their impacts on the environment.

The environmental variables that describe the abiotic part of the environment can be of different nature, depending for example on whether we study a terrestrial or an aquatic group of organisms. Typical groups of variables concern properties of the terrain (calculated from a digital elevation model), such as elevation, slope and exposition; geological composition of the terrain or the riverbed/seabed; physical and chemical properties of the soil/water/air, such as moisture, pH, quantities of pollutants, and so on. An important group of variables concerns climate and encompasses temperature, precipitation, etc.

Biological aspects of the environment that are considered in habitat models are typically more specific

and more directly related to the target group of organisms as compared to the abiotic variables. They may be rather coarse and refer to the community, e.g., when modeling brown bear habitat one of the inputs may be the type of forest at a particular location. They may also refer to more specific types of organisms that are related to the target group, e.g., when modeling the habitat of wolves, information on important prey species such as hare and deer may be taken into account.

Some environmental variables may involve both abiotic and biotic aspects. Land cover is a typical example: possible values for this variable may be forest, grassland, water, etc. Finally, some environmental variables are related to human activity: examples are proximity to settlements, population density, and proximity to roads/railways.

The output of a habitat model is some property of the population of the target group of organisms at the spatial unit of analysis. There are two degrees of freedom here: one stems from the target property, the other from the group of organisms studied. In the simplest case, the output is just the presence/absence of a single species (or group). In this case, we simply talk about habitat models.

An example habitat model for brown bears in Slovenia (taken from Jerina et al. 2003) is given in Table 19.1. It has the form of an IF-THEN rule, which specifies the conditions that define suitable habitat for brown bears. The rule uses three environmental variables PREDOMINANT-LAND-COVER, FOREST-ABUNDANCE and PROXIMITY-TO-SETTLEMENTS: it was actually learned by applying machine learning techniques to observational data.

We can also be interested in the abundance or density of the population. If we take these as indicators of the suitability of the environment for the group of organisms studied, we talk about habitat suitability models: the output of these models can be interpreted as a degree of suitability. The abundance of the population can be measured in terms of the number of individuals or their total size (e.g., the dry biomass of a certain species of algae). If the (taxonomic) group is large enough, we can also consider the diversity of the group (Shannon index, species richness or such like, see Krebs 1989).

Table 19.1 A habitat model for the brown bear (*Ursus arctos*) in Slovenia

IF	PREDOMINANT-LAND-COVER = Forest
AND	FOREST-ABUNDANCE > 60%
AND	PROXIMITY-TO-SETTLEMENTS > 1.5 km
THEN	BrownBearHabitat = Suitable
ELSE	BrownBearHabitat = Unsuitable

In the most general case of habitat modeling, we are interested in the relation between the environmental variables and the structure of the population at the spatial unit of analysis (absolute and relative abundances of the organisms in the group studied). One approach to this is to build habitat models for each of the organisms (or lower taxonomic units) in the group, then aggregate the outputs of these models to determine the structure of the population (or the desired target property). An alternative approach is to build a model that simultaneously predicts the presence/abundance of all organisms in the group or directly the desired target property of the entire group. A comparison of the two approaches in the context of machine learning of habitat models is given by Demšar et al. (2006a).

We should note here that observing the presence or absence of a species/group (or its abundance/density) within a given spatial unit can be a nontrivial task. While most plants and certain animals (such as sea cucumbers) are relatively immobile, many animals (including brown bears) can move fast and cover wide spatial areas. In the latter cases, one might consider areals of activity (home ranges) and sample from these to obtain data for learning habitat suitability models: this is what was done in the study by Jerina et al. (2003).

Another issue that commonly occurs in habitat modeling, especially in the context of machine learning, is the fact that only presence data are often collected (i.e., no absence data are usually available). In such cases, additional care is necessary when preparing the data for the modeling task. Examples (spatial units) where the target group can be reasonably expected not to occur (based on domain knowledge) may be considered as absence data.

Finally, let us reiterate that habitat modeling focuses on the spatial aspects of the distribution and abundance of plants and animals. It studies the relationships between some environmental variables and the presence/abundance of plants and animals, under

the implicit assumption that both are observed at a single point in time for a given spatial unit. It mostly ignores the temporal aspects of the distribution/abundance, the latter being the focus of population dynamics modeling. Still, some temporal aspects may be taken into account, for example, averages of environmental variables over a period of time preceding the observation are sometimes included in habitat models (e.g., average winter air temperature).

19.3 Machine Learning for Habitat Modeling

19.3.1 The Machine Learning Task of Predictive Modeling

The input to a machine learning algorithm is most commonly a single flat table comprising a number of fields (columns) and records (rows). In general, each row represents an object and each column represents a property (of the object). In machine learning terminology, rows are called examples and columns are called attributes (or sometimes features). Attributes that have numeric (real) values are called continuous attributes. Attributes that have nominal values (are called discrete attributes.

The tasks of classification and regression are the two most commonly addressed tasks in machine learning. They are concerned with predicting the value of one field from the values of other fields. The target field is called the class (dependent variable in statistical terminology). The other fields are called attributes (independent variables in statistical terminology).

If the class is continuous, the task at hand is called regression. If the class is discrete (it has a finite set of nominal values), the task at hand is called classification. In both cases, a set of data (dataset) is taken as input, and a predictive model is generated. This model can then be used to predict values of the class for new data. The common term predictive modeling refers to both classification and regression.

Given a set of data (a table), only a part of it is typically used to generate (induce, learn) a predictive model. This part is referred to as the training set. The remaining part is reserved for evaluating the predictive performance of the learned model and is called the testing set. The testing set is used to estimate the performance of the model on unseen data (and sometimes also called validation set, see Chapter 2 of this volume by Marzbahn).

More reliable estimates of performance on unseen data are obtained by using cross-validation, which partitions the entire data available into N (with N typically set to 10) subsets of roughly equal size. Each of these subsets is in turn used as a testing set, with all of the remaining data used as a training set. The performance figures for each of the testing sets are averaged to obtain an overall estimate of the performance on unseen data.

19.3.2 A Machine Learning Formulation of Habitat Modeling

In the case of habitat modeling, examples correspond to spatial units of analysis. The attributes correspond to environmental variables describing the spatial units, as these are the inputs to a habitat model. The class is a target property of the given (taxonomic) group of organisms, such as presence, abundance or diversity.

The habitat model from Table 19.1 has been learned from a dataset which includes the discrete attribute PREDOMINANT-LAND-COVER (which can have the value forest, among others) and the continuous attributes FOREST-ABUNDANCE and PROXIMITY-TO-SETTLEMENTS. The class BrownBear-Habitat is discrete, with Suitable and Unsuitable as possible values. Hence, we are dealing with a classification task. An excerpt from the dataset is given in Table 19.2.

The machine learning task of habitat modeling is thus defined as follows. Given is a set of data with rows corresponding to spatial locations (units of analysis), attributes corresponding to environmental variables, and the class corresponding to a target property of the population studied. The goal is to learn a predictive model that predicts the target property from the environmental variables (from the given dataset). If we are only looking at presence/absence or suitable/unsuitable as values of the class (as is the case above), we have a classification problem. If we are looking at the degree of suitability (density/abundance), we have a regression problem.

Table 19.2 An excerpt from the dataset for modeling brown bear habitat in Slovenia. PLC stands for PREDOMINANT-LAND-COVER, PTS for PROXIMITY-TO-SETTLEMENTS, and BBH for BrownBearHabitat

Location	PLC	FOREST-ABUNDANCE	PTS	OtherEnvVariables	BBH
l1	Forest	80	21.4	–	Yes
l2	Forest	66	13.9	–	Yes
l3	Forest	55	50.0	–	No
l4	Forest	72	1.2	–	No
l5	Grassland	6	19.1	–	No
l6	Grassland	0	11.4	–	No
l7	Wetland	3	5.8	–	No
l8	Water	0	3.9	–	No

19.3.3 Decision Tree Induction

19.3.3.1 What Are Decision Trees?

Decision trees (Breiman et al. 1984, see also Chapter 4 of this volume by Dattatreya) are hierarchical structures, where each internal node contains a test on an attribute, each branch corresponds to an outcome of the test, and each leaf node gives a prediction for the value of the class variable. Depending on whether we are dealing with a classification or a regression problem, the decision tree is called a classification or a regression tree, respectively. An example classification tree modeling the habitat of sea cucumbers is given in Fig. 19.1. The tree has been derived from actual data by using machine learning (Džeroski and Drumm 2003).

Regression tree leaves contain constant values as predictions for the class value. They thus represent piece-wise constant functions. Model trees, where leaf nodes can contain linear models predicting the class value, represent piece-wise linear functions. An example model tree that predicts the total abundance of hemi- and eu-edaphic Collembola is given in Fig. 19.2 (Kampichler et al. 2000).

Note that decision trees represent total partitions of the data space, where each test corresponds to an axis-parallel split. Most algorithms for decision tree induction consider axis-parallel splits. However, there are a few algorithms that consider splits along lines that need not be axis-parallel or even consider splits along non-linear curves.

19.3.3.2 Top-Down Induction of Decision Trees

Finding the smallest decision tree that would fit a given data set is known to be computationally expensive (NP-hard). Heuristic search, typically greedy, is thus employed to build decision trees. The common way to induce decision trees is the so-called Top-Down Induction of Decision Trees (TDIDT, Quinlan 1986). Tree construction proceeds recursively starting with the entire set of training examples (entire

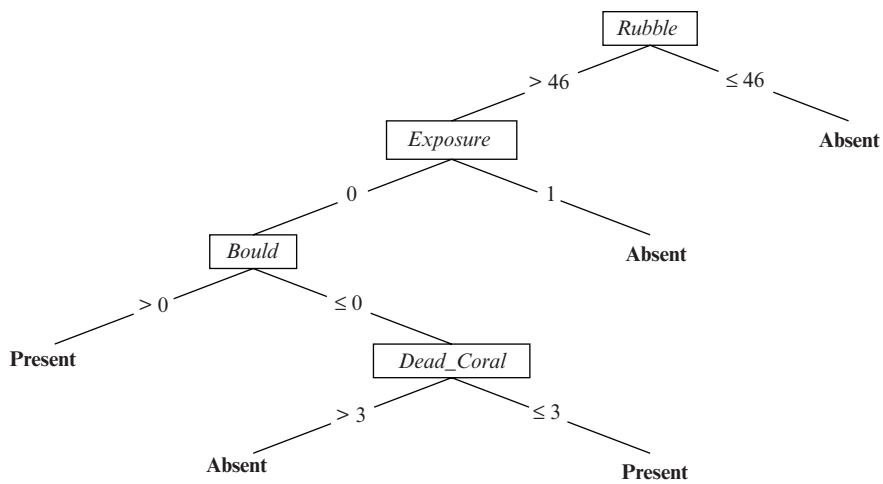


Fig. 19.1 A classification tree that predicts the suitability of habitat for the sea cucumber species *Holothuria Leucospilota* on Rarotonga, Cook Islands

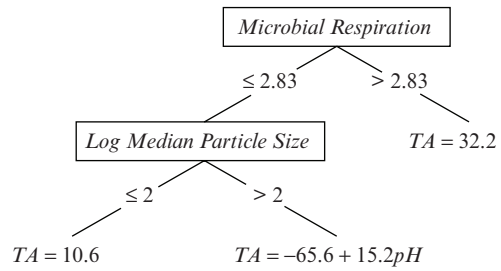


Fig. 19.2 A model tree that predicts the total abundance (TA) of hemi- and eu-edaphic Collembola on the FAM experimental farm at Scheyern (near Munich), Germany

table). At each step, an attribute is selected as the root of the (sub)tree and the current training set is split into subsets according to the values of the selected attribute.

For discrete attributes, a branch of the tree is typically created for each possible value of the attribute. For continuous attributes, a threshold is selected and two branches are created based on that threshold. For the subsets of training examples in each branch, the tree construction algorithm is called recursively. Tree construction stops when the examples in a node are sufficiently pure (i.e., all are of the same class) or if some other stopping criterion is satisfied (e.g., there is no good attribute to add at that point). Such nodes are called leaves and are labeled with the corresponding values of the class.

Different measures can be used to select an attribute in the attribute selection step. These also depend on whether we are inducing classification or regression trees (Breiman et al. 1984). For classification, Quinlan (1986) uses information gain, which is the expected reduction in entropy of the class value caused by knowing the value of the given attribute. Other attribute selection measures, however, such as the Gini index (Breiman et al. 1984) or the accuracy of the majority class, can and have been used in classification tree induction. In regression tree induction, the expected reduction in variance of the class value can be used.

An important mechanism used to prevent trees from over-fitting data is tree pruning. Pruning can be employed during tree construction (pre-pruning) or after the tree has been constructed (post-pruning). Typically, a minimum number of examples in branches can be prescribed for pre-pruning and a confidence level in accuracy estimates for leaves for post-pruning.

19.3.4 Rule Induction

19.3.4.1 What Are Predictive Rules?

We will use the word rule here to denote patterns of the form "IF Conjunction of conditions THEN Conclusion." The individual conditions in the conjunction will be tests concerning the values of individual attributes, such as "PROXIMITY-TO-SETTLEMENTS > 1.5 km" or "PREDOMINANT-LAND-COVER=Forest". For predictive rules, the conclusion gives a prediction for the value of the target (class) variable.

If we are dealing with a classification problem, the conclusion assigns one of the possible discrete values to the class, e.g., "BrownBearHabitat=Unsuitable". A rule applies to an example if the conjunction of conditions on the attributes is satisfied by the particular values of the attributes in the given example. Each rule corresponds to a hyper-rectangle in the data space.

Predictive rules can be ordered or unordered. Unordered rules are considered independently and several of them may apply to a new example that we need to classify. A conflict resolution mechanism is needed if two rules which recommend different classes apply to the same example. A default rule typically exists, whose recommendation is taken if no other rule applies.

Ordered rules form a so-called decision list. Rules in the list are considered from the top to the bottom of the list. The first rule that applies to a given example is used to predict its class value. Again, a default rule with an empty precondition is typically found as the last rule in the decision list and is applied to an example when no other rule applies.

An ordered list of rules describing brown bear habitat is given in Table 19.1: the second rule in this list is the default rule which always applies. An unordered list of rules that predicts the suitability of habitat for sea cucumbers is given in Table 19.3. Note that classification trees can be transcribed into sets of classification rules, since each of the leaves of a classification tree corresponds to a classification rule. Although less

Table 19.3 A set of unordered rules that predicts the suitability of habitat for the sea cucumber species *Holothuria Leucospilota* on Rarotonga, Cook Islands. The default rule, which predicts the class Absent is not listed

IF Sand < 7.5 AND Rubble > 62.0 AND Rock_Pave < 15.0 AND Dead_Coral < 13.5 THEN Presence = Present [3 absent, 15 present]
IF Rubble < 54.0 AND 7.5 < Consol_Rubble < 77.5 AND Bould < 25.0 AND Rock_Pave < 30.0 AND Dead_Coral < 45.0 THEN Presence = Present [1 absent, 6 present]
IF Rubble < 9.5 AND Live_Coral < 27.5 THEN Presence = Absent [65 absent]
IF Sand > 8.5 AND Consol_Rubble < 5.0 THEN Presence = Absent [64 absent]
IF Bould > 2.5 AND Rock_Pave > 30.0 THEN Presence = Absent [10 absent]

common in practice, regression rules also exist, and can be derived, e.g., by transcribing regression trees into rules.

19.3.4.2 The Covering Algorithm for Rule Induction

In the simplest case of binary classification, one of the classes is referred to as positive and the other as negative. For a classification problem with several class values, a set of rules is constructed for each class. When rules for class c_i are constructed, examples of this class are referred to as positive, and examples from all the other classes as negative.

The covering algorithm works as follows. We first construct a rule that correctly classifies some examples. We then remove the examples covered by the rule from the training set and repeat the process until no more examples remain. When learning ordered rules we remove all examples covered and when learning unordered rules only the positive examples covered by the rule.

Within this outer loop, different approaches can be taken to find individual rules. One approach is to heuristically search the space of possible rules top-down, i.e., from general to specific (in terms of examples covered this means from rules covering many to rules covering fewer examples) (Clark and Boswell 1991). To construct a single rule that classifies examples into class c_i, we start with a rule with an empty antecedent (IF part) and the selected class c_i as a consequent (THEN part). The antecedent of this rule is satisfied by all examples in the training set, and not only those of the selected class. We then progressively refine the antecedent by adding conditions to it, until only examples of class c_i satisfy the antecedent. To allow for handling imperfect data, we may construct a set of rules which is imprecise, i.e., does not classify all examples in the training set correctly.

19.4 Case Studies of Habitat Modeling with Machine Learning

In this section, we exemplify the machine learning approach to habitat modeling through four case studies. For each case study, we briefly describe the data available, the machine learning approach used, and the results obtained. We also give examples of habitat models learned in the process.

19.4.1 Bioindicator Organisms in Slovenian Rivers

In this study (Džeroski et al. 1997), we learned habitat models for 17 organisms that can be found in Slovenian rivers and are used as indicator organisms when determining the biological quality of river waters. The habitat models explicate the influence of physical and chemical parameters of river water on 10 plant taxa and seven animal taxa. On the plant side, eight kinds of diatoms (BACILLARIOPHYTA) and two kinds of green algae (CHLOROPHYTA) were studied. The animal taxa chosen for study include worms (OLIGOCHAETA), crustaceans (AMPHIPODA) and five kinds of insects.

The plant taxa studied were: *Coconeis placentula*, *Cymbella sp.*, *Cymbella ventricosa*, *Diatoma vulgare*, *Navicula cryptocephala*, *Navicula gracilis*, *Nitzschia palea*, *Synedra ulna*, *Cladophora sp.*, and *Oedogonium sp.* The animal taxa studied were *Tubifex sp.*,

Table 19.4 Example rules from the habitat models for bioindicator organisms in Slovenian rivers (*Nitzschia palea*, *Elmis sp.*, and *Plecoptera leuctra sp.*)

IF Hardness > 11.85 AND NO_2 > 0.095 AND NH_4 > 0.09 THEN Nitzschia = Present	IF NO_2 < 0.005 AND NO_3 < 7.1 AND PO_4 < 0.125 AND Detergents < 0.055 AND BOD < 2 THEN Nitzschia = Absent
IF Temperature > 12.75 AND BOD < 0.65 THEN Elmis = Present	IF PH > 7.05 AND BOD > 12.15 THEN Elmis = Absent
IF Temperature < 23 AND 120 < Saturation < 150 AND COD > 10.9 AND BOD < 3.75 THEN Leuctra = Present	IF Temperature < 22.25 AND Total Hardness < 18.55 AND BOD > 6.9 THEN Leuctra = Absent

Gammarus fossarum, *Baetis sp.*, *Leuctra sp.*, *Chironomidae (green)*, *Simulium sp.*, *Elmis sp.*

The data used in the study came from the Hydrometeorological Institute of Slovenia (now Environment Agency of Slovenia) that performs regular water quality monitoring for most Slovenian rivers and maintains a database of water quality samples. The data used cover a 4 year period, from 1990 to 1993. In total, 698 water samples were available on which both physical/chemical and biological analyses were performed: the former provided the environmental variables for the habitat models, while the latter provided information on the presence/absence of the studied organisms.

Plants are more or less influenced by the following physical and chemical parameters (water properties): total hardness, nitrogen compounds (NO_2 , NO_3, NH_4), phosphorus compounds (PO_4), silica (SiO_2), iron (Fe), surfactants (detergents), chemical oxygen demand (COD), and biochemical oxygen demand (BOD). The last two parameters indicate the degree of organic pollution: the first reflects the total amount of degradable organic matter, while the second reflects the amount of biologically degradable matter. Animals are mostly influenced by a different set of parameters: water temperature, acidity or alkalinity (pH), dissolved oxygen (O_2, saturation of O_2), total hardness, chemical (COD), and biochemical oxygen demand (BOD).

The habitat models for the plant/animal taxa used the following environmental variables: Hardness, NO_2, NO_3, NH_4, PO_4, SiO_2, Fe, Detergents, COD, BOD for plants and Temperature, PH, O_2, Saturation, COD, BOD for animals. The class is the presence of the selected taxon (with values Present and Absent). Seventeen machine learning problems were thus defined, one for each taxon. Each of the datasets contained 698 examples.

Rule induction, and in particular the CN2 system (Clark and Boswell 1991), was used to construct the habitat models. The rules induced on the complete data were given to a domain expert (river ecologist) for inspection. Their accuracy on unseen data was also estimated by dividing the data into a training set (70%) and a testing set (30%), repeating this 10 times and averaging the results (accuracy on the test set).

The accuracy of the 17 models on the whole (training) dataset ranges between 66% and 85%, while the default accuracy, i.e., the majority class frequency ranges from 50% to 70%. The estimated accuracy on unseen cases ranges from 53% to 71%. In nine of the 17 cases, the models substantially improve upon the default accuracy and provide interesting knowledge about the taxa studied.

In several cases, the induced rules are consistent with and confirm the expert knowledge about the organism studied. The diatom *Nitzschia palea*, the most common species in Slovenian rivers, is very tolerant to pollution. The rules confirm that a larger degree of pollution is beneficial to this species: they indicate that *Nitzschia palea* needs nitrogen compounds, phosphates, silica, and larger amounts of degradable matter (COD and BOD). *Elmis sp.* is known to inhabit clean waters: the rules demand a low quantity of biodegradable matter (pollution) in order for the taxon to be present, and predict that the taxon will be absent if the

water is overly polluted (has high values of BOD, COD and pH).

Not all of the induced rules agree with existing expert knowledge. For example, the rules that predict the presence of the taxon *Plecoptera leuctra sp.*, which is used as an indicator of clean waters, confirm that it is indeed found mainly in clean waters. However, they also state that it can be found in quite polluted water, provided there is enough oxygen. Thus, they enhance current knowledge on the bioindicator role of this taxon.

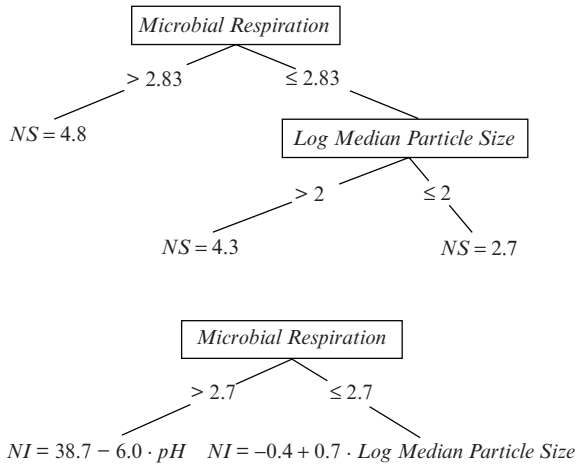


Fig. 19.3 A regression (top) and a model tree (bottom) that predict the number of species (NS) of hemi- and eu-edaphic Collembola and the number of individuals (NI) of the collembolan species *Folsomia quadrioculata*, respectively, on the FAM experimental farm at Scheyern (near Munich), Germany

19.4.2 Soil Insects on an Experimental Farm in Germany

Kampichler et al. (2000) used machine learning techniques to build habitat models for Collembola (springtails), the most abundant insects in soil, in an agricultural soil environment. They study both the taxonomic group of Collembola, as well as the dominant species in the study area, (*Folsomia quadrioculata*). The habitat models constructed relate the total abundance and species number of Collembola, as well as the abundance of the dominant species, to habitat characteristics, i.e., properties of the soil.

The data used in the study come from an experimental farm at Scheyern (near Munich), Germany, run by the FAM Research Network on Agroecosystems. The farm was of size approximately 153 ha, located at an elevation of 450–490 m above sea level, with mean annual temperature and mean annual precipitation of 7.58°C and 833 mm, respectively. In April 1991, one soil core was taken at each intersection of a 50 × 50 m mesh-size grid (7.8 cm diameter, 5 cm depth) and yielded a total of 396 cores. The majority of these points were situated in arable fields, the remainder in pastures, meadows and arable fields on former hop fields. Microarthropods were counted and Collembola identified by species. Only data of euedaphic (soil-dwelling) Collembola and hemiedaphic Collembola (which live near the soil surface) were included in the analysis.

To measure environmental factors, cores were taken from the same sampling points, at a distance of approximately 25 cm from the first cores. The following environmental variables were measured: microbial biomass, microbial respiration, soil moisture, soil acidity, carbon content (Ct) and nitrogen content (Nt). Soil texture at the sampling points was also determined and expressed by the (base 10) logarithm of the median particle size (diameter). From the 396 cores, only those that had no missing values for any of these variables were included in the model development, leaving a dataset of n = 195 samples.

To build habitat models, we used regression trees. More specifically, the system M5 (Quinlan 1992) for model tree induction was used. Trees were built separately for each of the three target variables: the abundance and diversity (species number) of Collembola, and the abundance of the dominant species *Folsomia quadrioculata*. Example trees for the last two are given in Fig. 19.3, while an example tree for the first is given in Fig. 19.2. Linear regression models, as well as neural networks with one hidden layer, were also constructed for each of the target variables.

In terms of predictive power, model trees fared better than linear regression and worse than neural networks. All of them, however, had quite low predictive power (for unseen cases, the correlation coefficients were estimated by 10-fold cross-validation at approx. 0.3 for linear regression, 0.4 for model trees and 0.5 for neural networks). The most probable reason for the low performance is that the aggregated spatial

distribution of collembolans sets limit to the possibility of predicting the actual number of collembolans. In this context, the quality of trees of being transparent and providing explicit information about the quantitative relationships between the variables proved very appealing to the domain experts.

The trees clearly identify microbial respiration as the most important factor influencing the collembolan community, followed by soil texture and soil acidity. The same environmental variables seem to be important for all three target variables and the structure of the individual trees is very similar. In this case, simultaneous prediction of all target variables seems reasonable: this can be done by applying predictive clustering trees (Blockeel et al. 1998), a generalized version of decision trees. This methodology, also called multi-objective classification/prediction, has been applied to habitat modeling for river communities (Džeroski et al. 2001) and for soil insects, including mites and springtails (Demšar et al. 2006b).

19.4.3 Brown Bears in Slovenia

The brown bear (*Ursus arctos*) occurs today in only a small part of its historical range: Slovenia is among the few European countries with a preserved viable indigenous brown bear population, as well as populations of other large predator species, such as wolf and lynx. The Slovenian bear population is a part of the continuous Alps-Dinaric-Pindos population: its core habitat (the forests of Kočevska and Snežnik in South-Western Slovenia) is connected with Gorski Kotar in Croatia in a unified block of bear habitat. This bear population is important also because it represents the source for natural re-colonization or reintroduction of the bear into Slovenia's neighboring countries Austria and Italy.

In their study, Jerina et al. (2003) address three aspects of the brown bear population in Slovenia: its size (and its evolution over time), its spatial expansion out of the core area, and its potential habitat based on natural habitat suitability. The results of the study include estimates of population size, a picture of the spatial expansion of the population and maps of its optimal and maximal potential habitat (based on natural suitability). All of these are relevant to the management of the Slovenian brown bear population. In this section, we summarize the habitat modeling aspect of the study.

The habitat models built were based on bear sightings data acquired in the last decade of the 20th century by the Hunters association of Slovenia, as well as data from a previous radio-tracking project. Since we were interested in the optimal habitat, best represented by females with cubs, we selected only such sightings. Instead of using a cloud of sighting location points as the basis for the models, we used an estimation of the inhabited area (IA) constructed by a kernel method: this method gives as output the frequency/probability with which individual points in space are occupied by brown bears.

The spatial unit of analysis was a pixel of size 500×500 m. Positive examples were sampled from the inhabited area. Examples for the "optimal" habitat model were sampled from areas that exceeded a high threshold of the probability of bear occupancy: This threshold was lower when sampling positive examples for the "maximal" potential habitat model. Negative examples were randomly sampled from the rest of the study area (i.e., not the IA), which presumably is less (or not at all) suitable for bear habitat.

The explanatory environmental variables were derived from several GIS (Geographical Information Systems) layers. These included land cover data, forest inventory data, settlements map, road map, and a digital elevation model. Example variables include forest abundance and proximity to settlements. A value of each of these variables was associated with each 500×500 m pixel. The method of decision tree induction, and in particular the See5 commercial product, based on the C4.5 (Quinlan 1993) algorithm, was used to build the "optimal" and "maximal" habitat models.

The decision tree for optimal habitat (Fig. 19.4a) takes into account the surrounding forest matrix size, forest abundance in each pixel, predominant land cover type, sub-regional density of human population, and the predominant forest association within each forest pixel. The decision tree for maximal habitat (Fig. 19.4b) is much simpler and only takes into account the predominant land cover type, forest abundance, and proximity to settlement. Note that the classification rule for predicting "maximal bear habitat", given in Table 19.1, is obtained by rewriting the tree in Fig. 19.4b.

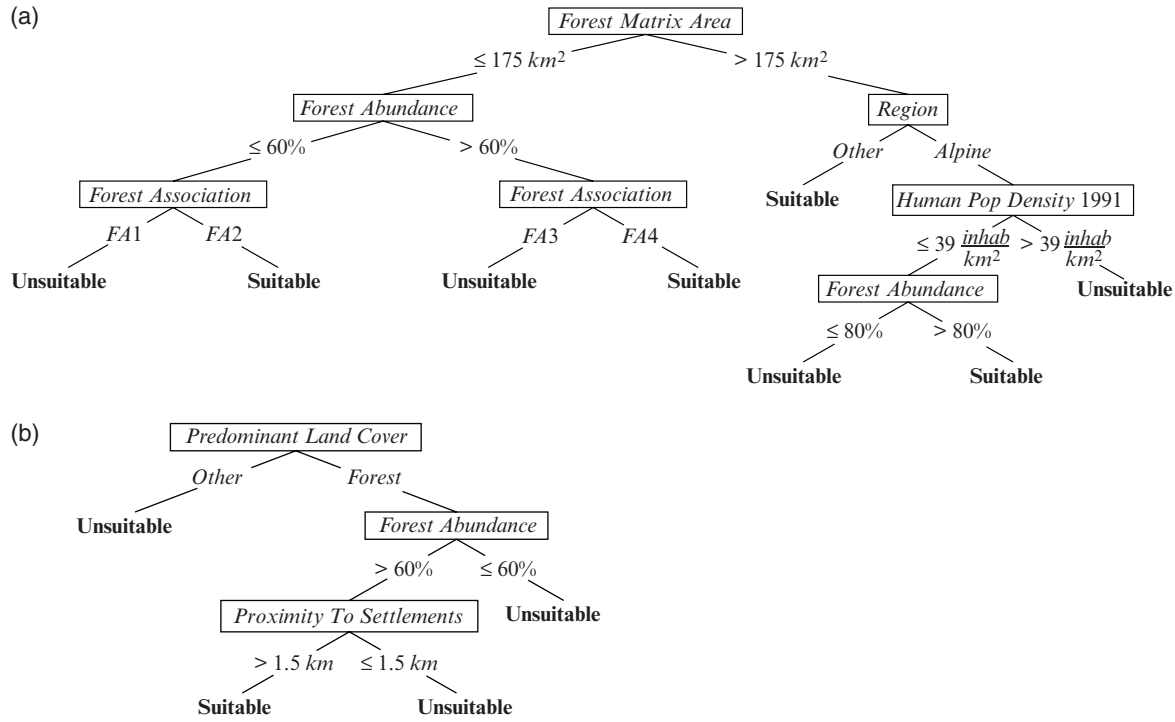


Fig. 19.4 Two decision trees predicting the (a) optimal and (b) maximal habitat of the brown bear (*Ursus arctos*) in Slovenia. FA1, FA2, FA3, and FA4 denote four different groups of forest associations, where FA1 and FA3 contain oak and FA2 and FA4 contain beech

The learned trees were used to produce the respective habitat maps. The thematic accuracy for the first map was estimated by 10-fold cross-validation as 89%, and 84% for the latter. The optimal habitat covers 12.3% of Slovenias territory, mostly in the southern part, bordering to Croatia. The possible maximal habitat extent includes additional 26.4% of the territory, mostly in the alpine region in the northern and western part of Slovenia, thus totaling 38.7% of the country.

It can be gleaned both from the decision trees as well as from the final habitat maps, that the bear habitat suitability in Slovenia largely depends on the presence of a dense forest cover, while it depends less upon food availability. Considering the increasing trend of forest cover in Slovenia, and assuming a continuation of high reproduction rates, we could even expect a further expansion of bear-inhabited areas in the future. It is furthermore obvious that the six-lane Ljubljana Trieste highway cuts through the optimal habitat at two vulnerable bottlenecks, disrupting the dispersion corridors towards the Alps: This can be seen from a large number of bear related traffic accidents on the highway. The habitat maps we constructed were used to recommend suitable locations for eco-ducts (wildlife bridges) across this highway to the Highway authority of Slovenia.

19.4.4 Sea Cucumbers on Rarotonga, Cook Islands

In the Pacific Islands, invertebrates including sea cucumbers are among the most valuable and vulnerable inshore fisheries resources. The sea cucumber (*Holothuria leucospilota*) forms an important part of the traditional subsistence fishery on Rarotonga, Cook Islands, yet little is known of this species present spatial distribution and abundance around the island. To contribute to the knowledge about this species, Džeroski and Drumm (2003) apply machine learning to measured data and build a habitat model that predicts the number of sea cucumber

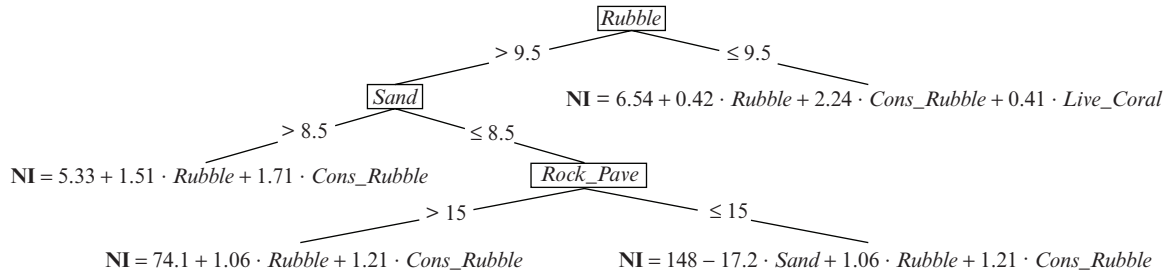


Fig. 19.5 A model tree predicting the number of individuals (NI) of the sea cucumber species *H. Leucospilota* in a 2 × 50 m transect of the sea bed near Rarotonga, Cook Islands

individuals from environmental characteristics of a location.

The spatial unit of analysis was a 2 × 50 m (100 m^2) strip transect: This size was selected to account for the patchy distribution of the animals. A total of 128 sites were sampled for environmental and biological variables. The number of *H. leucospilota* individuals encountered along each transect was recorded. In addition to the species abundance, 10 environmental variables that were expected to have an influence on the habitat preference of the sea cucumber were recorded. These included the exposure of the site (windward or leeward side of the island), and the following microhabitat variables, estimated as a percentage (with possible values from 0% to 100%) of the total 100 m^2 area sampled: Sand, Rubble, Cons_Rubble (consolidated rubble), Boulder, reef rock/pavement (Rock_Pave), live coral (Live_Coral), dead coral (Dead_Coral), mud/silt (Mud_Silt), and Gravel.

The number of *H. leucospilota* individuals was the class variable, while the 10 environmental variables were the attributes. Model tree induction was used to build the habitat model. More specifically, M5', a re-implementation of the system M5 (Quinlan 1992) within the software package WEKA (Witten and Frank 1999) was used. The model tree constructed is given in Fig. 19.5. The correlation coefficient for predictions on unseen cases was estimated to be 0.5 (by using 10-fold cross-validation).

The tree identifies the most important influences of the site characteristics on habitat suitability (rubble and sand, followed by rock pavement, consolidated rubble, and live coral). It identifies four types of sites (one leaf for each) and constructs different linear models to predict the number of sea cucumbers at each.

Two of the site types are essentially not very suitable as sea cucumber habitat: the first (LM1) does not have enough rubble, while the second (LM4) does have enough rubble, but also has too much sand. The average numbers of individuals recorded at the two types of sites are 15 and 35, respectively. One site type (LM2) is very suitable as sea cucumber habitat, as evidenced by the average of 236 animals found per site. This type of site is characterized by enough rubble, little sand and little rock pavement. The last type of site (LM3) represents a moderately suitable habitat for sea cucumbers: it has the same characteristics as the most suitable habitat, except for too much rock pavement. The sea cucumbers prefer larger percentages of rubble and consolidated rubble in all four types of sites (positive coefficients for rubble/consolidated rubble in each of the four linear models).

19.5 Summary and Discussion

In this chapter, we have introduced the task of habitat suitability modeling and formulated it as a machine learning problem. Habitat-suitability modeling studies the effect of the abiotic characteristics of the habitat on the presence, abundance or diversity of a given taxonomic group of organisms. We have briefly described two approaches to machine learning that are often used in habitat suitability modeling: decision tree induction and rule induction.

Applications of machine learning to habitat suitability modeling can be grouped along two dimensions. One dimension is the type of environment where

the studied group of organisms lives, e.g., aquatic (river or sea) or terrestrial (forest or agricultural fields). Another dimension is the type of machine learning approach used, e.g., symbolic (decision trees or classification rules) or statistical (logistic regression or neural networks).

In this chapter, we have given examples of using symbolic machine learning approaches to construct models of habitat suitability for several kinds of organisms in the abovementioned environments. These include habitat models for springtails and other soil organisms in an agricultural setting, brown bears in a forest environment, bioindicator organisms in a river environment, and finally sea cucumbers in a sustainable fishing setting. Many more examples of using machine learning for habitat modeling exist, some of which we point to below. A collection of papers, devoted specifically to the topic of habitat modeling, has been edited by Raven et al. (2002) and describes several applications of machine learning methods.

The author has been involved in quite a few other habitat modeling applications of machine learning, besides those summarized above. These include another, more realistic application in modeling the effects of agricultural actions on soil insects, including mites and collembolans (Demšar et al. 2006). This has been also studied in the context of farming with genetically modified crops and their effects on soil fauna, including earthworms (Debeljak et al. 2005). We have also studied habitat suitability for red deer in Slovenian forests using GIS data, such as elevation, slope, and forest composition (Debeljak et al. 2001).

Neural networks are often used for habitat modeling: several applications are described in (Lek and Guegan 1999). For example, (Lek-Ang et al. 1999) use them to study the influence of soil characteristics, such as soil temperature, water content, and proportion of mineral soil on the abundance and species richness of *Collembola* (springtails). Another study of habitat suitability modeling by neural networks is given by Ozesmi and Ozesmi (1999).

Several habitat-suitability modeling applications of other data mining methods are surveyed by Fielding (1999b). Fielding (1999a) applies a number of methods, including discriminant analysis, logistic regression, neural networks and genetic algorithms, to predict nesting sites for golden eagles. Bell (1999) uses decision trees to describe the winter habitat of pronghorn antelope. Jeffers (1999) uses a genetic algorithm to discover rules that describe habitat preferences for aquatic species in British rivers.

As compared to traditional statistical methods, such as linear and logistic regression, the use of machine learning offers several advantages. On one hand, machine learning methods are capable of approximating nonlinear relationships (typical for the interactions between living organisms and the environment) better than traditional linear approaches. On the other hand, symbolic learning approaches, such as decision trees and classification rules, provide understandable models that can be inspected to give insight into the domain studied.

Let us conclude by mentioning several recent research topics related to the use of machine learning for habitat suitability modeling. These include machine learning methods for simultaneous prediction of several target variables, machine learning methods that are spatially aware and finally the use of habitat suitability modeling in the context of predicting the effects of climate change. We discuss each of these briefly below.

When modeling the habitat of a group of organisms, we are interested in the relation between the environmental variables and the structure of the population at the spatial unit of analysis (absolute and relative abundances of the organisms in the group studied). While one approach to this is to build habitat models for each of the organisms, the alternative approach of building a model that simultaneously predicts the presence/abundance of all organisms in the group is more natural. For this purpose, we can use a neural network with several output nodes that share a common hidden layer. Recently, however, symbolic machine learning approaches have been developed that address this problem, namely predictive clustering trees (Blockeel et al. 1998) and predictive clustering rules (Ženko et al. 2006) for multi-target prediction.

When using machine learning to build habitat models, individual spatial points are treated as training examples. These are assumed to be completely independent and their relative spatial position (proximity) is ignored. This can result in unrealistic predictions of very small patches of habitat: this was, e.g., the case in the brown bear habitat modeling study described

earlier in the chapter. This problem is usually dealt in a post-processing phase, where the prediction of the habitat model for each spatial unit are corrected by taking into account (the predictions for) the neighborhood of that unit. However, spatially aware machine learning methods have recently started to emerge (Lee et al. 2005, Andrienko et al. 2005), although applications of such methods in habitat modeling are still rare.

Finally, let us mention climate change, which is already causing significant changes in the distribution of animals and vegetation across the globe. Predicting future effects along these lines is an emerging area where the use of machine learning for habitat modeling is likely to increase drastically. The idea in this context is to build habitat models for the target groups of organisms, which include climate-related variables, such as mean annual temperature and precipitation. By applying the habitat models to the predictions produced by climate models, one can predict the changes of the distribution of the target group of organisms. For example, Ogris and Jurc (2007) study the change of potential habitats for different tree species under varying climate change scenarios.

Harrison et al. (2006) conduct a more global study where the changes of habitat are investigated for a much larger and more diverse group of organisms. In their study, the availability of suitable climate space across Europe for the distributions of 47 species was modelled. These were chosen to encompass a range of taxa (including plants, insects, birds and mammals) and to reflect dominant and threatened species from 10 habitats. Habitat availability was modelled for the current climate and three climate change scenarios using a neural network model, showing that the distribution of many species in Europe may be affected by climate change, but that the effects are likely to differ between species.

In sum, machine learning methods have been successfully used and are increasingly more often used for habitat modeling, establishing the relations between abiotic characteristics of the environment and the properties of a target population of organisms (such as presence, abundance or diversity). The learned models can be used as tools for the management of the population studied. Perhaps even more importantly, the learned model can enhance our knowledge of the studied population.

References

Allaby, M. (1996). *Basics of environmental science*. London: Routledge.

Andrienko, G., Malerba, D., May, M., & Teisseire, M. (Eds.) (2005). *Proceedings of the ECML/PKDD 2005 Workshop on Mining Spatio-Temporal Data*. Portugal: University of Porto.

Bell, J. F. (1999). Tree based methods. In A. H. Fielding, (Ed.) *Machine learning methods for ecological applications* (pp. 89–105). Dordrecht: Kluwer.

Blockeel, H., De Raedt, L., & Ramon, J. (1998). Top-down induction of clustering trees. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 55–63). San Francisco: Morgan Kaufmann.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth.

Clark, P., & Boswell, R. (1991). Rule induction with CN2: Some recent improvements. *Proceedings of the Fifth European Working Session on Learning* (pp. 151–163). Berlin: Springer.

Debeljak, M., Džeroski, S., Jerina, K., Kobler, A., & Adamič, M. (2001). Habitat suitability modelling of red deer (Cervus elaphus, L.) in South-Central Slovenia. *Ecological Modelling*, *138*, 321–330.

Debeljak, M., Cortet, J., Demšar, D., & Džeroski, S. (2005). Using data mining to assess the effects of Bt maize on soil microarthropods. *Proceedings of the Nineteenth International Conference Informatics for Environmental Protection* (pp. 615-620). Czech Republic: Brno.

Demšar, D., Džeroski, S., Debeljak, M., & Henning Krogh, P. (2006a). Predicting aggregate properties of soil communities vs. community structure in an agricultural setting. *Proceedings of the Twentieth International Conference on Informatics for Environmental Protection* (pp. 295–302). Aachen: Shaker Verlag.

Demšar, D., Džeroski, S., Larsen, T., Struyf, J., Axelsen, J., Bruns-Pedersen, M., & Henning Krogh, P. (2006b). Using multi-objective classification to model communities of soil microarthropods. *Ecological Modelling*, *194*, 131–143.

Džeroski, S. (2001). Applications of symbolic machine learning to ecological modelling. *Ecological Modelling*, *146*, 263-273, 2001.

Džeroski, S., & Drumm, D. (2003). Using regression trees to identify the habitat preference of the sea cucumber (Holothuria leucospilota) on Rarotonga, Cook Island. *Ecological Modelling, 170*, 219–226.

Džeroski, S., Grbović, J., & Blockeel, H. (2001). Predicting river water communities with logical decision trees. *Book of Abstracts, Third European Ecological Modelling Conference*, Croatia: Dubrovnik.

Džeroski, S., Grbović, J., Walley, W. J., & Kompare, B. (1997). Using machine learning techniques in the construction of models. Part II: Rule induction. *Ecological Modelling*, *95*, 95–111.

Fielding, A. H. (1999a). An introduction to machine learning methods. In A. H. Fielding, (Ed.) *Machine learning methods for ecological applications* (pp. 1–35). Dordrecht: Kluwer.

Fielding, A. H. (Ed.) (1999b). *Machine learning methods for ecological applications*. Dordrecht: Kluwer.

Harrison, P. A., Berry, P. M., Butt, N., & New, M. (2006). Modelling climate change impacts on species distributions at the European scale: implications for conservation policy. *Environmental Science and Policy*, *9*(2). 116–128.

Jeffers, J. N. R. (1999). Genetic algorithms I. In A. H. Fielding (Ed.) , *Machine learning methods for ecological applications* (pp. 107–121). Dordrecht: Kluwer.

Jerina, K., Debeljak, M., Kobler, A., & Adamič, M. (2003). Modeling the brown bear population in Slovenia: A tool in the conservation management of a threatened species. *Ecological Modelling*, *170*, 453-469.

Joergensen, S. E., & Bendoricchio, G. (2001). *Fundamentals of ecological modelling*. Amsterdam: Elsevier.

Kampichler, C., Džeroski, S., & Wieland, R. (2000). The application of machine learning techniques to the analysis of soil ecological data bases: Relationships between habitat features and *Collembola* community characteristics. *Soil Biology and Biochemistry 32*, 197–209.

Krebs, C. J. (1972). *Ecology*. New York: Harper and Row.

Krebs, C. J. (1989). *Ecological methodology*. HarperCollins, New York, NY.

Lee, C.-H., Greiner, R., & Schmidt, M. (2005). Support vector random fields for spatial classification. *Proceedings of the Ninth European Conference on Principles and Practice of Knowledge Discovery in Databases* (pp. 121–132). Berlin: Springer.

Lek-Ang, S., Deharveng, L., & Lek, S. (1999). Predictive models of collembolan diversity and abundance in a riparian habitat. *Ecological Modelling*, *120*(2-3), 247–260.

Lek, S., & Guegan, J. F. Guest (Eds.) (1999). Application of artificial neural networks in ecological modelling. Special issue of *Ecological Modelling 120*(2-3).

Mitchell, T. (1997). *Machine learning*. New York: McGraw-Hill.

Ogris, N., & Jurc, M. (2007). Potential changes in the distribution of maple species (*Acer pseudoplatanus, A. campestre, A. platanoides, A. obtusatum*) due to climate change in Slovenia. *Proceedings of the Symposium on Climate Change Influences on Forests and Forestry*. Slovenia: University of Ljubljana.

Ozesmi, S. L., & Ozesmi, U. (1999). An artificial neural network approach to spatial habitat modelling with interspecific interaction. *Ecological Modelling*, *116*(1), 15–31.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Quinlan J. R. (1992). Learning with continuous classes. *Proceedings of the Fifth Australian Joint Conference on Artificial Intelligence* (pp. 343–348). Singapore: World Scientific.

Quinlan, J. R. (1993). *Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Raven, P. H., Scott, J. M., Heglund, P., & Morrison, M. (2002). *Predicting species occurrences: Issues of accuracy and scale*. Washington, DC: Island Press.

Witten, I. H., & Frank, E. (1999). *Data mining: Practical machine learning tools and techniques with Java implementations*. San Francisco: Morgan Kaufmann.

Ženko, B., Džeroski, S., & Struyf, J. (2006). Learning predictive clustering rules. *Proceedings of the Fourth International Workshop on Knowledge Discovery in Inductive Databases* (pp. 234–250). Berlin: Springer.

Glossary

All-Frame Procedure: see **leave-one-out procedure**.

Artificial Intelligence: The science and engineering of designing models of systems based directly on data.

Apparent Error: The value of a performance measure when a model trained on some data set is evaluated on that same data set. Also called training error.

Artificial Neural Networks: A branch of computational intelligence that is closely related to machine learning. It is loosely related to modeling cortical structures of the brain. An interconnected network of processing nodes, each of which contains a linear regression model, in many cases with its output modified by a nonlinear mapping function.

Auto-associative Neural Network: A neural network model where the output targets are the same as the inputs.

Bellman Optimality Equation: An equation that characterizes the optimal value function, and, indirectly, the optimal policies for an MDP.

Binary Genetic Algorithm: A genetic algorithm in which the variables to be optimized are encoded in binary form.

Binary Tree: A tree which has exactly two child nodes for every non-leaf node.

Blending: A method of crossover in a continuous genetic algorithm in which all variables become a linear combination of the mother value and the father value.

Bootstrap: A parameter estimation and model selection method wherein multiple **bootstrap samples** are drawn from a single data set. It is also used for estimating the prediction error. See **resampling methods**, and **cross-validation**.

Bootstrap Sample: A sample drawn, with replacement, from a data set, and with the same size as the data set.

Cardinal Variable: A variable that takes values over an uncountable ordered set.

Child-Node: For a decision tree, a node following another node in a path from root node to a leaf node.

Chromosome: For a genetic algorithm, a concatenation of the genes representing the variables to be optimized into a potential solution.

Classification: The process or model relating a set of variables to a set of discrete variables, with the values of the discrete variable denoting distinct classes of objects. Also called Discrimination. Compare with **regression**.

Clear-Air Turbulence: Turbulence in the free atmosphere that is not caused by thunderstorms or wind flow over terrain.

Confidence: In fuzzy logic algorithms, a value that represents the quality and/or relevance of a quantity (e.g., a measurement) as a membership value between 0 and 1.

Continuous Variable Genetic Algorithm: A genetic algorithm in which the variables to be optimized are encoded as real values.

Computational Intelligence: The branch of artificial intelligence based on heuristic algorithms.

Cost Function: The function to be minimized in an optimization problem. Also known as objective function, or error function, and the negative of the fitness function.

Crossover: The process of chromosomes exchanging information during the mating stages of a genetic algorithm.

Crossover Point: The point in a genetic algorithm chromosome where the crossover occurs. Also called a **kinetochore**.

Crossover Rate: The percentage of the population of chromosomes that will mate at each generation in a genetic algorithm.

Cross-Validation: A model selection technique wherein a single sample data set is partitioned into multiple segments; some segments are used for training a model, and the resulting model is tested on the unused segments. It can also be used for estimating the prediction error. Compare with **bootstrapping**. See **training set** and **validation set**.

Cycle: A path in a graph with the same beginning and ending vertex.

Decision Boundary: The line, surface, or hypersurface between observation sets for different decisions in a graph.

Decision Region: The set of all possible observation points assigned to a particular decision.

Decision Tree: A tree with a rule for deciding the child node to traverse to, at every non-leaf node.

Degree of a Node: The number of edges that the node touches in a graph.

Developmental Data: The dataset explored by the AI algorithm in search of exploitable relationships. Like **training set**. Compare to **validation set**.

Directed Acyclic Graph (DAG): A directed graph with no cycles.

Directed Graph: A graph with only ordered pairs of nodes for edges.

Discounted Problem (DCP): An MDP in which the discount factor, α, is less than 1.

Dynamic Programming: A field in engineering and mathematics that addresses how to find optimal controls in dynamical systems subject to a degree of randomness.

Edge or Link: A pair of vertices in a graph.

Elitism: In a genetic algorithm, the process of maintaining the lowest cost solution in the population unchanged until it is supplanted.

Expert System: An algorithm built to emulate the reasoning of a human expert in solving a specific problem.

Feature Space: The set of all possible observations of all available measurement variables. Also known as a **solution space**.

Feed-Forward Neural Network: A neural network model where the nodes (or neurons) are connected only between layers of otherwise unconnected nodes. The flow of information is generally in one direction – from input nodes toward output nodes.

Fitness: The level of "goodness" of a genetic algorithm chromosome as judged by the **fitness function**.

Fitness Function: The function to be maximized in an optimization problem. The negative of the **cost function** and related to the **objective function**.

Forward Model: A model whose calculations, from input to output, move in the same direction as physical causality.

Fuzzy Consensus Method: An "ensemble of experts" in which multiple predictions of a variable of interest are averaged using weights based on relevance and confidence values.

Fuzzy Logic: A generalization of classical logic that provides a framework for reasoning with fuzzy sets.

Fuzzy Set: In contrast to classical or "crisp" sets, fuzzy sets represent ambiguous or natural language concepts by assigning each element a degree of set membership between 0 and 1.

Gene: The smallest unit of information in a genetic algorithm. Depending on the encoding used, each variable to be optimized may be represented by one or more genes. Genes are concatenated into **chromosomes**.

Generalization Error: Same as **prediction error**.

Generation: A full iteration of the genetic algorithm, including evaluation of the cost function, natural selection, mate selection, mating, and mutation.

Genetic Algorithm (GA): A technique in search and optimization that is based on the principals of genetic combination and evolution of populations. The GA is very robust and useful for finding global minima.

Global Minimum: The lowest value of the cost function in the entire solution space.

Global Optimization: The process of finding the most optimal (minimum or maximum) value of an objective function in the entire solution space.

Graph: A set of vertices and a set of pairs of vertices chosen from the vertex set.

Graph Traversal: Systematically visiting nodes in a graph.

Graphical Turbulence Guidance (GTG): An operational fuzzy-logic algorithm that produces forecasts of atmospheric turbulence for aviation users.

Height of a Rooted Tree: For a decision tree, the length of the longest path of all the paths from the root node to all leaf nodes.

Hidden Node: An artificial (unobserved) variable in a neural network.

Hybrid Genetic Algorithm: A combination of a genetic algorithm, which finds the basin of the global minimum, with a more traditional optimization method (often a gradient descent method) to rapidly find the lowest cost function value in that region of the solution space.

Information Criterion: A criterion used to select the best model among a number of models with different complexities.

Inverse Model: A model whose calculations, from input to output, move in the opposite direction as physical causality.

Inverse Problem: Solving for the input of a forward model given known values of the output.

Kinetochore: The point in a genetic algorithm chromosome where the crossover occurs. Also called **crossover point**.

Labeled Training Sample: A data vector with a known decision.

Leaf Node: In a decision tree, a non-root node with a degree of one in a tree. Also known as a terminal node.

Leave-One-Out Procedure: A cross-validation method. Given a training set of N input-target patterns, by leave-one-out procedure one can train the network N times, each time leaving out one pattern from training, and just using it to compute an error criterion. Also called all-frame procedure.

Length of a Path: The number of edges in the path of a graph.

Linear Discriminant Function: A linear function of predictors, separating different classes of objects according to some criterion.

Linear Model: A parametric model, linear in the parameters, but not necessarily linear in the relationship between variables.

Local Minimum: In optimization, a minimum point in the cost function in a region of the solution space that is not the global minimum.

Machine Learning: A broad subfield of artificial intelligence that is concerned with the development of algorithms and techniques that allow computer programs to "learn".

Mating: The process of combining the variables from parent chromosomes to form offspring chromosomes in a genetic algorithm.

Mate Selection: In a genetic algorithm, the process of determining which chromosomes will mate.

Mamdami-Style Fuzzy Inference: A fuzzy logic method for reasoning to a "crisp" conclusion based on several fuzzy rules.

Markov Decision Process (MDP): In reinforcement learning, a problem in which a learning agent interacts with an environment and at each timestep experiences state transitions and feedback (costs or rewards) that are a (possibly random) function only of the state and the action taken, and not the previous history.

Membership Function: The fuzzy analog of a characteristic function, representing the degree to which elements belong to a fuzzy set as real numbers between 0 and 1.

Model Selection: The task of selecting "the best" model from a set of models, based on some criterion. Compare with **model averaging**.

Model Averaging: The task of averaging over the outputs of a set of models, for the purpose of arriving at a single output. Compare with **model selection**.

Multilayered Perceptron: A **feed-forward neural network** with multiple layers of nodes.

Mutation: The process of randomly changing some of the genes in a genetic algorithm chromosome.

Mutation Rate: The percentage of genes that are mutated in a genetic algorithm.

Natural Selection: In a genetic algorithm, the process of eliminating chromosomes from the population that are less fit, that is, the value of their cost function is not as optimal as that of other members of the population.

NCAR Improved Moments Algorithm (NIMA): A fuzzy logic algorithm for analyzing Doppler spectra (e.g., from radar) and producing moment estimates and associated confidences.

Neural Networks: See **artificial neural networks**.

Nonparametric Model: A model capable of representing any function, even when it is written in a parametric form.

Nominal Variable: A variable that takes values over a finite unordered set.

Objective Function: The function to be optimized in an optimization problem. Related to **cost function** and **fitness function**.

Offspring: A chromosome that results from a mating operation in a genetic algorithm.

Optimization: The process of finding the best set of variable values to meet the objective of the problem.

Ordinal Variable: A variable that takes values over a countable ordered set.

Out-Degree: The number of edges originating from a node of a graph.

Output: The predicted value for a **target**. See also **training**.

Overfitting: The phenomenon of fitting a relatively small data set with a model that has an abundance of parameters. Unless the model is expected to be a "look-up

table" representing the data in all its details, overfitting generally leads to a reduction in predictive strength.

Parent: A genetic algorithm chromosome chosen for the mating operation.

Partially Observable MDP (POMDP): In reinforcement learning, a problem in which the learning agent does not know the environmental state of an underlying **MDP** exactly.

Partition: A set of mutually exclusive and collectively exhaustive subsets.

Path: A sequence of vertices with each overlapping pair of successive vertices being a valid edge in a graph.

Pattern Classification: The assignment of a data vector to one of a known set of classes.

Parametric Model: A model incapable of representing all functions. Usually written in a parametric form, but not always. Also see **nonparametric model**.

Perceptron: A **feed-forward neural network** with no hidden layer of nodes (i.e., with one hidden layer of weights).

Policy: In reinforcement learning, a deterministic or stochastic (probabilistic) set of rules that specify what action to choose given knowledge of the environmental state.

Population (in Genetic Algorithms): A set of chromosomes in a genetic algorithm, each of which represents a potential solution to the optimization problem.

Population (in Statistics): A set of objects not all of which are available for observation or measurement.

Population Size (in Genetic Algorithms): The number of chromosomes in the population of a genetic algorithm.

Prediction Error: The expected value of a performance measure if a model were to be applied to a population. Given the nature of a population (in statistics), this error is estimated from a sample, often via some resampling technique.

Principal Component Analysis: A data analysis method that extracts from a high dimensional dataset (i.e., with many variables) a smaller number of principal variables (called principal components), which capture the main variance in the dataset.

***Q*-Learning:** A reinforcement learning algorithm that learns the optimal Q-value for an **MDP**, and hence an optimal policy, based on interaction with an environment.

***Q*-Value:** In reinforcement learning, a function that for each environmental state and each possible action gives the discounted sum of future costs expected when starting in that state with that action and then following a specified policy.

Radar Echo Classifier (REC): Three related fuzzy logic algorithms for classifying Doppler weather radar returns: the Anomalous-Propagation ground clutter Detection Algorithm (REC-APDA), the Precipitation Detection Algorithm (REC-PDA) and the Sea Clutter Detection Algorithm (REC-SCDA).

Regression: The process or model relating a set of variables to a set of continuous variables. Compare with **classification**.

Regularization: A method of preventing an AI algorithm from overfitting the developmental data as it tunes modeled relationship. Regularization becomes important when solving ill-conditioned inverse problems.

Reinforcement Learning: A field of study that addresses how a learning agent's experience of feedback from an environment can be used to develop an optimal policy, or control. Also known as stochastic dynamic programming.

Resampling Methods: A set of methods (including **cross-validation** and **bootstrap**) wherein a single sample data set is partitioned into multiple sets. These methods are used for model selection, and for arriving at unbiased estimates of some quantity.

Retrieval: See **inverse problem**.

Rooted Directed Acyclic Graph (RDAG): DAG with a root node from which paths exist to all other nodes.

Rooted Tree: A tree structure with one vertex identified as the root node.

Roulette Wheel Selection: In a genetic algorithm, a form of mate selection in which the pairing of parent chromosomes is achieved via random selection. Can be accomplished either with or without rank or cost weighting.

Sample: A subset of a population (in statistics).

Self-Organizing Map (SOM): A method for clustering data onto a flexible mesh. This can be regarded as a discrete form of nonlinear principal component analysis.

Single Point Crossover: In a genetic algorithm, combining the information from two parents by exchanging the portion of the chromosome to the right of the kinetochore in parent one with that of parent two.

Selection: The process of choosing the parents to be mated in a genetic algorithm.

Solution Space: The space spanned by all possible solutions to an optimization problem.

Stochastic Shortest Path Problem (SSPP): An MDP in which the discount factor, α, is equal to 1 and there is a "final" state (usually denoted as state 0) in which the process terminates.

Takagi-Sugeno Fuzzy Inference: A fuzzy logic method for estimating a variable of interest based on two or more fuzzy rules.

Target: A measured quantity which is to be predicted. Also called the response, or the predictand. Compare with **output**.

Temporal Difference Algorithm: In reinforcement learning, a method for updating an estimate for an **MDP**'s value function based on successive estimates of the value of a state as new experience is obtained.

Test Set: A data set not used in training, nor in assessing model complexity. It is usually used for arriving at an unbiased estimate of prediction error without any **resampling**.

Training: The process of estimating the parameters of a model, usually designed to maximally align **output**(s) of the model with **target** values. Also called parameter estimation in statistics.

Training Error: Same as **apparent error**.

Training Set: A data set employed for estimating parameters of a model.

Tree: An **undirected graph** with no cycles.

Tournament Selection: In a genetic algorithm, pairing parent chromosomes for mating by randomly choosing a number of random chromosomes, then selecting for mating those chromosomes of the selected sub-population with the best cost function value.

Undirected Graph: A graph with only unordered pairs of nodes for edges.

Uniform Crossover: In a genetic algorithm, combining the information from two parents by generating a random mask of 1's and 0's that determine whether each gene comes from parent 1 or parent 2. For a continuous variable GA, the combination could include blending the genes in random complementary fractions.

Unsupervised Classification: Grouping a set of data vectors with no prior information about the distinguishing nature of different groups, i.e., without a target. Also called cluster analysis in statistics.

Validation Set: A data set employed for either selecting a model or assessing its performance, when the set is sufficiently large. The validation data set should be independent of the training data set from which the relationship was derived.

Value Function: In reinforcement learning, a function that for each environmental state gives the discounted sum of future costs expected when starting in that state and following a specified policy.

Vertex or Node: In a graph, any physical or abstract entity with possible relations to other such entities.

Weights: Parameters representing the strength of the relationship between two variables. The variables may be either observed or not (e.g., hidden nodes).

Index