Undergraduate Topics in Computer Science

M. Narasimha Murty
V. Susheela Devi

Pattern Recognition

An Algorithmic Approach

Springer
Universities Press
UTiCS

Undergraduate Topics in Computer Science

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

For further volumes:
http://www.springer.com/series/7592

M. Narasimha Murty · V. Susheela Devi

Pattern Recognition

An Algorithmic Approach

Prof. M. Narasimha Murty
Indian Institute of Science
Dept. of Computer Science
and Automation
Bangalore
India
mnm@csa.iisc.ernet.in

Dr. V. Susheela Devi
Indian Institute of Science
Dept. of Computer Science
and Automation
Bangalore
India
susheela@csa.iisc.ernet.in

A co-publication with the Universities Press (India) Pvt. Ltd, licensed for sale in all countries outside of India, Pakistan, Bhutan, Bangladesh, Sri Lanka, Nepal, The Maldives, Middle East, Malaysia, Indonesia and Singapore. Sold and distributed within these territories by the Universities Press (India) Private Ltd.

ISSN 1863-7310
ISBN 978-0-85729-494-4 e-ISBN 978-0-85729-495-1
DOI 10.1007/978-0-85729-495-1
Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2011922756

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Contents

Preface *xi*

1. Introduction **1**

1.1 What is Pattern Recognition? 2
1.2 Data Sets for Pattern Recognition 4
1.3 Different Paradigms for Pattern Recognition 4
Discussion 5
Further Reading 5
Exercises 6
Bibliography 6

2. Representation **7**

2.1 Data Structures for Pattern Representation 8
2.1.1 Patterns as Vectors 8
2.1.2 Patterns as Strings 9
2.1.3 Logical Descriptions 9
2.1.4 Fuzzy and Rough Pattern Sets 10
2.1.5 Patterns as Trees and Graphs 10
2.2 Representation of Clusters 15
2.3 Proximity Measures 16
2.3.1 Distance Measure 16
2.3.2 Weighted Distance Measure 17
2.3.3 Non-Metric Similarity Function 18
2.3.4 Edit Distance 19
2.3.5 Mutual Neighbourhood Distance (MND) 20
2.3.6 Conceptual Cohesiveness 22
2.3.7 Kernel Functions 22
2.4 Size of Patterns 23
2.4.1 Normalisation of Data 23
2.4.2 Use of Appropriate Similarity Measures 24
2.5 Abstractions of the Data Set 24
2.6 Feature Extraction 26

2.6.1 Fisher's Linear Discriminant 26
2.6.2 Principal Component Analysis (PCA) 29
2.7 Feature Selection 31
2.7.1 Exhaustive Search 32
2.7.2 Branch and Bound Search 33
2.7.3 Selection of Best Individual Features 36
2.7.4 Sequential Selection 36
2.7.5 Sequential Floating Search 37
2.7.6 Max–Min Approach to Feature Selection 39
2.7.7 Stochastic Search Techniques 41
2.7.8 Artificial Neural Networks 41
2.8 Evaluation of Classifiers 41
2.9 Evaluation of Clustering 43
Discussion 43
Further Reading 44
Exercises 44
Computer Exercises 46
Bibliography 46

3. Nearest Neighbour Based Classifiers 48

3.1 Nearest Neighbour Algorithm 48
3.2 Variants of the NN Algorithm 50
3.2.1 k-Nearest Neighbour (kNN) Algorithm 50
3.2.2 Modified k-Nearest Neighbour (MkNN) Algorithm 51
3.2.3 Fuzzy kNN Algorithm 53
3.2.4 r Near Neighbours 54
3.3 Use of the Nearest Neighbour Algorithm for Transaction Databases 54
3.4 Efficient Algorithms 55
3.4.1 The Branch and Bound Algorithm 56
3.4.2 The Cube Algorithm 58
3.4.3 Searching for the Nearest Neighbour by Projection 60
3.4.4 Ordered Partitions 62
3.4.5 Incremental Nearest Neighbour Search 64
3.5 Data Reduction 65
3.6 Prototype Selection 66
3.6.1 Minimal Distance Classifier (MDC) 61
3.6.2 Condensation Algorithms 69
3.6.3 Editing Algorithms 75
3.6.4 Clustering Methods 77
3.6.5 Other Methods 79

Discussion 79
Further Reading 79
Exercises 80
Computer Exercises 82
Bibliography 83

4. Bayes Classifier 86

4.1 Bayes Theorem 86
4.2 Minimum Error Rate Classifier 88
4.3 Estimation of Probabilities 90
4.4 Comparison with the NNC 91
4.5 Naive Bayes Classifier 93
4.5.1 Classification using Naive Bayes Classifier 93
4.5.2 The Naive Bayes Probabilistic Model 93
4.5.3 Parameter Estimation 95
4.5.4 Constructing a Classifier from the Probability Model 96
4.6 Bayesian Belief Network 97
Discussion 100
Further Reading 100
Exercises 100
Computer Exercises 101
Bibliography 102

5. Hidden Markov Models 103

5.1 Markov Models for Classification 104
5.2 Hidden Markov Models 111
5.2.1 HMM Parameters 113
5.2.2 Learning HMMs 113
5.3 Classification Using HMMs 116
5.3.1 Classification of Test Patterns 116
Discussion 118
Further Reading 119
Exercises 119
Computer Exercises 121
Bibliography 122

6. Decision Trees 123

6.1 Introduction 123
6.2 Decision Trees for Pattern Classification 125
6.3 Construction of Decision Trees 131

6.3.1 Measures of Impurity 132
6.3.2 Which Attribute to Choose? 133
6.4 Splitting at the Nodes 136
6.4.1 When to Stop Splitting 138
6.5 Overfitting and Pruning 139
6.5.1 Pruning by Finding Irrelevant Attributes 139
6.5.2 Use of Cross-Validation 140
6.6 Example of Decision Tree Induction 140
Discussion 143
Further Reading 143
Exercises 144
Computer Exercises 145
Bibliography 145

7. Support Vector Machines 147

7.1 Introduction 147
7.1.1 Linear Discriminant Functions 147
7.2 Learning the Linear Discriminant Function 154
7.2.1 Learning the Weight Vector 154
7.2.2 Multi-class Problems 156
7.2.3 Generality of Linear Discriminants 166
7.3 Neural Networks 169
7.3.1 Artificial Neuron 169
7.3.2 Feed-forward Network 171
7.3.3 Multilayer Perceptron 173
7.4 SVM for Classification 175
7.4.1 Linearly Separable Case 177
7.4.2 Non-linearly Separable Case 181
Discussion 183
Further Reading 183
Exercises 183
Computer Exercises 186
Bibliography 187

8. Combination of Classifiers 188

8.1 Introduction 188
8.2 Methods for Constructing Ensembles of Classifiers 190
8.2.1 Sub-sampling the Training Examples 190
8.2.2 Manipulating the Input Features 196

8.2.3 Manipulating the Output Targets 196
8.2.4 Injecting Randomness 197
8.3 Methods for Combining Classifiers 199
Discussion 202
Further Reading 203
Exercises 203
Computer Exercises 204
Bibliography 205

9. Clustering 207

9.1 Why is Clustering Important? 216
9.2 Hierarchical Algorithms 221
9.2.1 Divisive Clustering 222
9.2.2 Agglomerative Clustering 225
9.3 Partitional Clustering 229
9.3.1 k-Means Algorithm 229
9.3.2 Soft Partitioning 232
9.4 Clustering Large Data Sets 233
9.4.1 Possible Solutions 234
9.4.2 Incremental Clustering 235
9.4.3 Divide-and-Conquer Approach 236
Discussion 239
Further Reading 240
Exercises 240
Computer Exercises 242
Bibliography 243

10. Summary 245

11. An Application: Handwritten Digit Recognition 247

11.1 Description of the Digit Data 247
11.2 Pre-processing of Data 249
11.3 Classification Algorithms 249
11.4 Selection of Representative Patterns 249
11.5 Results 250
Discussion 254
Further Reading 254
Bibliography 254

Index 255

Preface

Our main aim in writing this book is to make the concepts of pattern recognition clear to undergraduate and postgraduate students of the subject. We will not deal with pre-processing of data. Rather, assuming that patterns are represented using some appropriate pre-processing techniques in the form of vectors of numbers, we will describe algorithms that exploit numeric data in order to make important decisions like classification. Plenty of worked out examples have been included in the book along with a number of exercises at the end of each chapter.

The book would also be of use to researchers in the area. Readers interested in furthering their knowledge of the subject can benefit from the ''Further Reading'' section and the bibliography given at the end of each chapter.

Pattern recognition has applications in almost every field of human endeavour including geology, geography, astronomy and psychology. More specifically, it is useful in bioinformatics, psychological analysis, biometrics and a host of other applications. We believe that this book will be useful to all researchers who need to apply pattern recognition techniques to solve their problems.

We thank the following students for their critical comments on parts of the book: Vikas Garg, Abhay Yadav, Sathish Reddy, Deepak Kumar, Naga Malleswara Rao, Saikrishna, Sharath Chandra, and Kalyan.

M Narasimha Murty
V Susheela Devi

1 Introduction

Learning Objectives

At the end of this chapter, you will

- Be able to define pattern recognition
- Understand its importance in various applications
- Be able to explain the two main paradigms for pattern recognition problems
  - Statistical pattern recognition
  - Syntactic pattern recognition

Pattern recognition can be defined as the classification of data based on knowledge already gained or on statistical information extracted from patterns and/or their representations.

It has several important applications. Multimedia document recognition (MDR) and automatic medical diagnosis are two such applications. For example, in MDR we have to deal with a combination of text, audio and video data. The text data may be made up of alpha-numerical characters corresponding to one or more natural languages. The audio data could be in the form of speech or music. Similarly, the video data could be a single image or a sequence of images—for example, the face of a criminal, his fingerprint and signature could come in the form of a single image. It is also possible to have a sequence of images of the same individual moving in an airport in the form of a video clip.

In a typical pattern recognition application, we need to process the raw data and convert it into a form that is amenable for a machine to use. For example, it may be possible to convert all forms of a multimedia data into a vector of feature values. The frequency of occurrence of important keywords in the text could form a part of the representation; the audio data could be represented based on linear predictive coding (LPC) coefficients and the video data could be represented in a transformed domain. Some of the popularly used transforms are the wavelet transform and the Fourier transform. Signal processing deals with the conversion of raw data into a vector of

numbers. This forms a part of the pre-processing of data. In this book, we will not deal with pre-processing of data.

Rather, assuming that patterns are represented using some appropriate pre-processing techniques in the form of vectors of numbers, we will describe algorithms that exploit the numeric data in order to make important decisions like classification. More specifically, we will deal with algorithms for pattern recognition. Pattern recognition involves classification and clustering of patterns. In classification, we assign an appropriate class label to a pattern based on an abstraction that is generated using a set of training patterns or domain knowledge. Clustering generates a partition of the data which helps decision making; the specific decision making activity of interest to us here is classification. For example, based on the multimedia data of an individual, in the form of fingerprints, face and speech, we can decide whether he is a criminal or not.

In the process, we do not need to deal with all the specific details present in the data. It is meaningful to summarise the data appropriately or look for an apt abstraction of the data. Such an abstraction is friendlier to both the human and the machine. For the human, it helps in comprehension and for the machine, it reduces the computational burden in the form of time and space requirements. Generating an abstraction from examples is a well-known paradigm in machine learning. Specifically, learning from examples or supervised learning and learning from observations or clustering are the two important machine learning paradigms that are useful here. In artificial intelligence, the machine learning activity is enriched with the help of domain knowledge; abstractions in the form of rule-based systems are popular in this context. In addition, data mining tools are useful when the set of training patterns is large. So, naturally pattern recognition overlaps with machine learning, artificial intelligence and data mining.

1.1 What is Pattern Recognition?

In pattern recognition, we assign labels to patterns. In Figure 1.1, there are patterns of Class X and Class O. Pattern P is a new sample which has to be assigned either to Class X or Class O. In a system that is built to classify humans into tall, medium and short, the abstractions, learnt from examples, facilitate assigning one of these class labels (''tall'', ''medium'' or ''short'') to a newly encountered human. Here, the class labels are semantic; they convey some meaning. In the case of clustering, we group a collection of unlabelled patterns; here, the labels assigned to each group of patterns are syntactic, or simply the cluster identity.

It is possible to directly use a classification rule without generating any abstraction. In such a case, the notion of proximity/similarity (or distance) is used to classify patterns. Such a similarity function is computed based on the representation of the

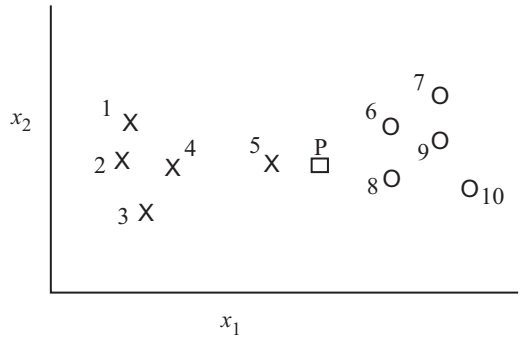


Figure 1.1 Example set of patterns

patterns; the representation scheme plays a crucial role is classification. A pattern is represented as a vector of feature values. The features which are used to represent patterns are important. For example, consider the data shown in Table 1.1 where humans are to be categorised into two groups ''tall'' and ''short''. The classes are represented using the feature ''weight''.

If a newly encountered person weighs 46 kg, then he/she may be assigned the class label ''short'' because 46 is closer to 50. However, such an assignment does not appeal to us because we know that the weight of an individual and the class labels ''tall'' and ''short'' do not correlate well; a feature such as ''height'' is more appropriate. Chapter 2 deals with representation of patterns and classes.

Table 1.1 Classification of humans into groups "tall" or "short" using the feature "weight"

Weight of human (in kilogrammes)	Class label
40	tall
50	short
60	tall
70	short

One of the important aspects of pattern recognition is its application potential. It has been used in a very wide variety of areas including agriculture, education, security, transportation, finance, medicine and entertainment. Specific applications are in the field of biometrics, bioinformatics, multimedia data analytics, document recognition, fault diagnostics, and expert systems. Classification is so fundamental an idea to humans that pattern recognition naturally finds a place in any field. Some of the most frequently cited applications include character recognition, speech/speaker recognition, and object recognition in images. Readers interested in some of these applications may refer to popular journals such as *Pattern Recognition (www.elsevier.com/locate/pr)* and *IEEE Transactions on Pattern Analysis and Machine Intelligence (www.computer.org/tpami)* for details.

1.2 Data Sets for Pattern Recognition

There are a wide variety of data sets available on the Internet. One popular site is the machine learning repository at UC Irvine (www.ics.uci.edu/MLRepository.html). It contains a number of data sets of varying sizes which can be used by any classification algorithm. Many of them even give the classification accuracy for some classification methods which can be used as a benchmark by researchers. Large data sets used for data mining tasks are available at kdd.ics.uci.edu and www.kdnuggets.com/datasets/.

1.3 Different Paradigms for Pattern Recognition

There are several paradigms which have been used to solve the pattern recognition problem. The two main ones are

1. Statistical pattern recognition
2. Syntactic pattern recognition

Of the two, statistical pattern recognition is more popular and has received the majority of attention in literature. The main reason for this is that most practical problems in this area deal with noisy data and uncertainty—statistics and probability are good tools to deal with such problems. On the other hand, formal language theory provides the background for syntactic pattern recognition and systems based on such linguistic tools are not ideally suited to deal with noisy environments. This naturally prompts us to orient material in this book towards statistical classification and clustering.

In statistical pattern recognition, we use vector spaces to represent patterns and classes. The abstractions typically deal with probability density/distributions of points in multi-dimensional spaces. Because of the vector space representation, it is meaningful to talk of sub-spaces/projections and similarity between points in terms of distance measures. There are several soft computing tools associated with this notion. For example, neural networks, fuzzy set and rough set based pattern recognition schemes employ vectorial representation of points and classes.

The most popular and simple classifier is based on the nearest neighbour rule. Here, a new pattern is classified based on the class label of its nearest neighbour. In such a classification scheme, we do not have a training phase. A detailed discussion on the nearest neighbour classification is presented in Chapter 3. It is important to look at the theoretical aspects of the limits of classifiers under uncertainty. The Bayes classifier characterises optimality in terms of minimum error rate classification. It is discussed in Chapter 4. The use of hidden Markov models (HMMs) is popular in fields like speech recognition. HMMs are discussed in Chapter 5. A decision tree is a transparent data structure which can deal with classification of patterns employing both numerical and categorical features. We discuss decision tree classifiers in Chapter 6.

Neural networks are modelled to mimic how the human brain learns. One type of neural network, the perceptron, is used to find the linear decision boundaries in high dimensional spaces. Support vector machines (SVMs) are built based on this notion. In Chapter 7, the role of neural networks and SVMs in classification is explored. It is meaningful to use more than one classifier to arrive at the class label of a new pattern. Such combinations of classifiers form the basis for Chapter 8.

Often, it is possible to have a large training data set which can be directly used for classification. In such a context, clustering can be used to generate abstractions of the data and these abstractions can be used for classification. For example, sets of patterns corresponding to different classes can be clustered to form sub-classes. Each such sub-class (cluster) can be represented by a single prototypical pattern; these representative patterns can be used to build the classifier instead of the entire data set. In Chapter 9, a discussion on some of the popular clustering algorithms is presented.

Discussion

Pattern recognition deals with classification and clustering of patterns. It is so fundamental an idea that it finds a place in many a field. Pattern recognition can be done statistically or syntactically—statistical pattern recognition is more popular as it can deal with noisy environments better.

Further Reading

Duda et al. (2000) have written an excellent book on pattern classification. The book by Tan et al. (2007) on data mining is a good source for students. Russell and Norvig (2003) have written a book on artificial intelligence which discusses learning and pattern recognition techniques as a part of artificial intelligence. Neural network as used for pattern classification is discussed by Bishop (2003).

Exercises

1. Consider the task of recognising the digits 0 to 9. Take a set of patterns generated by the computer. What could be the features used for this problem? Are the features semantic or syntactic? If the features chosen by you are only syntactic, can you think of a set of features where some features are syntactic and some are semantic?

2. Give an example of a method which uses a classification rule without using abstraction of the training data.
3. State which of the following directly use the classification rule and which uses an abstraction of the data.
   (a) Nearest neighbour based classifier
   (b) Decision tree classifier
   (c) Bayes classifier
   (d) Support vector machines
4. Specify whether the following use statistical or syntactic pattern recognition.
   (a) The patterns are vectors formed by the features. The nearest neighbour based classifier is used.
   (b) The patterns are complex and are composed of simple sub-patterns which are themselves built from simpler sub-patterns.
   (c) Support vector machines are used to classify the patterns.
   (d) The patterns can be viewed as sentences belonging to a language.

Bibliography

1. Bishop, C. M. *Neural Networks for Pattern Recognition.* New Delhi: Oxford University Press. 2003.
2. Duda, R. O., P. E. Hart and D. G. Stork. *Pattern Classification*. John Wiley and Sons. 2000.
3. Russell, S. and P. Norvig. *Artificial Intelligence: A Modern Approach.* Pearson India. 2003.
4. Tan, P. N., M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Pearson India. 2007.

2 Representation

Learning Objectives

After reading this chapter, you will

- Know that patterns can be represented as
  - Strings
  - Logical descriptions
  - Fuzzy and rough sets
  - Trees and graphs
- Have learnt how to classify patterns using proximity measures like
  - Distance measure
  - Non-metrics which include
    - k-median distance
    - Hausdorff distance
    - Edit distance
    - Mutual neighbourhood distance
    - Conceptual cohesiveness
    - Kernel functions
- Have found out what is involved in abstraction of data
- Have discovered the meaning of feature extraction
- Know the advantages of feature selection and the different approaches to it
- Know the parameters involved in evaluation of classifiers
- Understand the need to evaluate the clustering accomplished

A pattern is a physical object or an abstract notion. If we are talking about the classes of animals, then a description of an animal would be a pattern. If we are talking about various types of balls, then a description of a ball (which may include the size and material of the ball) is a pattern. These patterns are represented by a set of

descriptions. Depending on the classification problem, distinguishing features of the patterns are used. These features are called attributes. A pattern is the representation of an object by the values taken by the attributes. In the classification problem, we have a set of objects for which the values of the attributes are known. We have a set of classes and each object belongs to one of these classes. The classes for the case where the patterns are animals could be mammals, reptiles etc. In the case of the patterns being balls, the classes could be football, cricket ball, table tennis ball etc. Given a new pattern, the class of the pattern is to be determined. The choice of attributes and representation of patterns is a very important step in pattern classification. A good representation is one which makes use of discriminating attributes and also reduces the computational burden in pattern classification.

2.1 Data Structures for Pattern Representation

2.1.1 Patterns as Vectors

An obvious representation of a pattern will be a vector. Each element of the vector can represent one attribute of the pattern. The first element of the vector will contain the value of the first attribute for the pattern being considered. For example, if we are representing spherical objects, (30, 1) may represent a spherical object with 30 units of weight and 1 unit diameter. The class label can form part of the vector. If spherical objects belong to class 1, the vector would be (30, 1, 1), where the first element represents the weight of the object, the second element, the diameter of the object and the third element represents the class of the object.

EXAMPLE 1

Using the vector representation, a set of patterns, for example can be represented as

1.0, 1.0, 1 ; 1.0, 2.0, 1
2.0, 1.0, 1 ; 2.0, 2.0, 1
4.0, 1.0, 2 ; 5.0, 1.0, 2
4.0, 2.0, 2 ; 5.0, 2.0, 2
1.0, 4.0, 2 ; 1.0, 5.0, 2
2.0, 4.0, 2 ; 2.0, 5.0, 2
4.0, 4.0, 1 ; 5.0, 5.0, 1
4.0, 5.0, 1 ; 5.0, 4.0, 1

where the first element is the first feature, the second element is the second feature and the third element gives the class of the pattern. This can be represented graphically as shown in Figure 2.1, where patterns of class 1 are represented using the symbol +, patterns of class 2 are represented using X and the square represents a test pattern.

2.1.2 Patterns as Strings

The string may be viewed as a sentence in a language, for example, a DNA sequence or a protein sequence.

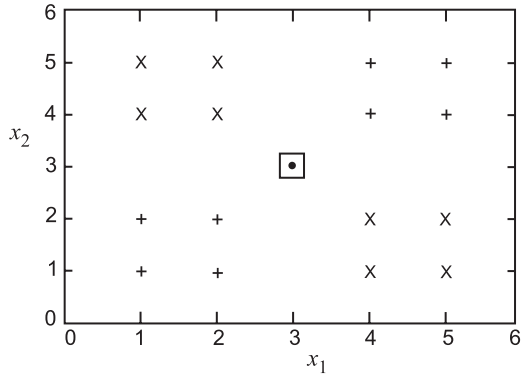


Figure 2.1 Example data set

As an illustration, a gene can be defined as a region of the chromosomal DNA constructed with four nitrogenous bases: adenine, guanine, cytosine and thymine, which are referred to by A, G, C and T respectively. The gene data is arranged in a sequence such as

GAAGTCCAG...

2.1.3 Logical Descriptions

Patterns can be represented as a logical description of the form

$$(x_1 = a_1..a_2) \wedge (x_2 = b_1..b_2) \wedge ...$$

where x_1 and x_2 are the attributes of the pattern and a_i and b_i are the values taken by the attribute.

This description actually consists of a conjunction of logical disjunctions. An example of this could be

$$(\text{colour} = \text{red} \vee \text{white}) \wedge (\text{make} = \text{leather}) \wedge (\text{shape} = \text{sphere})$$

to represent a cricket ball.

2.1.4 Fuzzy and Rough Pattern Sets

Fuzziness is used where it is not possible to make precise statements. It is therefore used to model subjective, incomplete and imprecise data. In a fuzzy set, the objects belong to the set depending on a membership value which varies from 0 to 1.

A rough set is a formal approximation of a crisp set in terms of a pair of sets which give the lower and the upper approximation of the original set. The lower and upper approximation sets themselves are crisp sets. The set X is thus represented by a tuple $\{\underline{X}, \overline{X}\}$ which is composed of the lower and upper approximation. The lower approximation of X is the collection of objects which can be classified with full certainty as members of the set X. Similarly, the upper approximation of X is the collection of objects that may possibly be classified as members of the set X.

The features of the fuzzy pattern may be a mixture of linguistic values, fuzzy numbers, intervals and real numbers. Each pattern X will be a vector consisting of linguistic values, fuzzy numbers, intervals and real values. For example, we may have linguistic knowledge such as ''If X_1 is small and X_2 is large, then class 3''. This would lead to the pattern (small, large) which has the class label 3. Fuzzy patterns can also be used in cases where there are uncertain or missing values. For example, the pattern maybe $X = (?, 6.2, 7)$. The missing value can be represented as an interval which includes its possible values. If the missing value in the above example lies in the interval [0,1], then the pattern can be represented as

$X = ([0, 1], 6.2, 7)$ with no missing values.

The values of the features may be rough values. Such feature vectors are called rough patterns. A rough value consists of an upper and a lower bound. A rough value can be used to effectively represent a range of values of the feature. For example, power may be represented as (230, 5.2, $(\underline{50}, \overline{49, 51})$), where the three features are voltage, current and frequency (represented by a lower and upper bound).

In some cases, hard class labels do not accurately reflect the nature of the available information. It may be that the pattern categories are ill-defined and best represented as fuzzy sets of patterns. Each training vector x_i is assigned a fuzzy label $u_i \in [0, 1]^c$ whose components are the grades of membership of that pattern to each class.

The classes to which the patterns belong may be fuzzy concepts, for example, the classes considered may be short, medium and tall.

2.1.5 Patterns as Trees and Graphs

Trees and graphs are popular data structures for representing patterns and pattern classes. Each node in the tree or graph may represent one or more patterns. The minimum spanning tree (MST), the Delauney tree (DT), the R-tree and the k-d tree are examples of this. The R-tree represents patterns in a tree structure

which splits space into hierarchically nested and possibly overlapping minimum bounding rectangles or bounding boxes. Each node of an R-tree has a number of entries. A non-leaf node stores a way of identifying the node and the bounding box of all entries of nodes which are its descendants. Some of the important operations on an R-tree are appropriate updation (insertion, deletion) of the tree to reflect the necessary changes and searching of the tree to locate the nearest neighbours of a given pattern. Insertion and deletion algorithms use the bounding boxes from the nodes to ensure that the nearby elements are placed in the same leaf node. Search entails using the bounding boxes to decide whether or not to search inside a node. In this way most of the nodes in the tree need not be searched.

A set of patterns can be represented as a graph or a tree where following a path in the tree gives one of the patterns in the set. The whole pattern set is represented as a single tree. An example of this is the frequent pattern (FP) tree.

Minimum Spanning Tree

Each pattern is represented as a point in space. These points can be connected to form a tree called the minimum spanning tree (MST). A tree which covers all the nodes in a graph is called a spanning tree. If $d(X, Y)$ is the distance or dissimilarity between nodes X and Y, an MST is a spanning tree where the sum of the distances of the links (edges in the tree) is the minimum. Figure 2.2 shows an example of a minimum spanning tree.

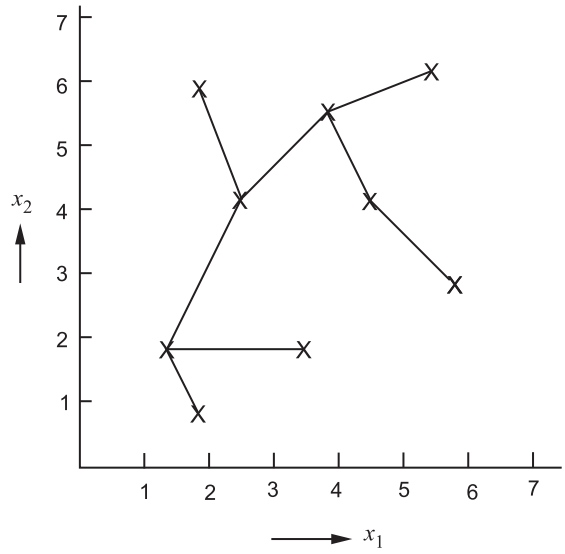


Figure 2.2 Example of a minimum spanning tree

The minimum spanning tree can be used for clustering data points. This is illustrated with the following example.

EXAMPLE 2

In Figure 2.3, 8 patterns are considered. Figure 2.4 gives the minimum spanning tree for the 8 patterns.

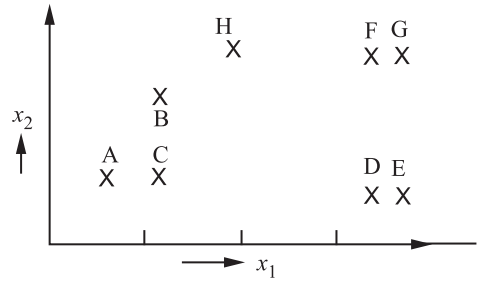


Figure 2.3 Patterns represented in feature space

The minimum spanning tree is used for clustering applications. The largest links in the MST are deleted to obtain the clusters. In Figure 2.4, if the largest link FD is deleted, it results in the formation of two clusters. The first cluster has points A, B, C, H, F and G, and the second cluster has the points D and E. In the first cluster, if the largest link HF is deleted, three clusters are formed. The first cluster has points A, B, C and H; the second cluster has points F and G; and the third cluster has points D and E.

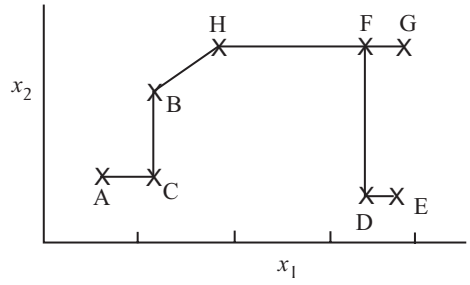


Figure 2.4 The MST for Figure 2.3

Frequent Pattern Trees

This data structure is basically used in transaction databases. The frequent pattern tree (FP tree) is generated from the transactions in the database. It is a compressed

tree structure which is useful in finding associations between items in a transaction database. This means that the presence of some items in a transaction will probably imply the presence of other items in the same transaction. The FP growth algorithm used for efficient mining of frequent item sets in large databases uses this data structure.

The first step in constructing this tree is to determine the frequency of every item in the database and sort them from largest to smallest frequencies. Then each entry in the database is ordered so that the order matches the frequency just calculated from largest to smallest. The root of the FP tree is first created and labelled null. The first transaction is taken and the first branch of the FP tree is constructed according to the ordered sequence. The second transaction is then added according to the ordering done. The common prefix shared by this transaction with the previous transaction will follow the existing path, only incrementing the count by one. For the remaining part of the transaction, new nodes are created. This is continued for the whole database. Thus it can be seen that the FP tree is a compressed tree which has information about the frequent items in the original database.

EXAMPLE 3

Consider a 4 $\times$ 4 square, where each of the squares is a pixel to represent a digit. The squares are assigned an alphabet as shown in Figure 2.5. For example, digit 4 would be represented in the 4 $\times$ 4 square as shown in Figure 2.6 and denoted by a, e, g, i, j, k, l, o. The digits 0, 1, 7, 9 and 4 can be represented as shown in Table 2.1.

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

Figure 2.5 Square representing the pixels of a digit

1			
1		1	
1	1	1	1
		1	

Figure 2.6 Square representing the pixels of digit 4

Table 2.1 A transaction database

Digit	Transaction
0	a, d, e, h, i, l, m, p, b, c, n, o
1	d, h, l, p
7	a, b, c, d, h, l, p
9	a, b, c, d, i, j, k, l, p, e, h
4	a, e, g, i, j, k, l, o

A scan of the transaction database in Table 2.1 will yield the frequency of every item which when sorted from largest to smallest gives (l : 5), (a : 4), (d: 4), (p: 4), (h: 3), (i: 3), (c: 3), (e: 3). Only items which have a frequency of 3 and above are listed here. Note that ties are resolved arbitrarily.

Table 2.2 The transaction database ordered according to frequency of items

Sl.No.	Transaction
0	l, a, d, p, h, i, c, e
1	l, d, p, h
7	l, a, d, p, h
9	l, a, d, p, i, e
4	l, a, i

Table 2.2 shows the transaction database ordered according to the frequency of items. Items which have a support below a threshold are weeded out. In this case, items which have support of two or below are left out. Thus, e, m, b, c, j, k, g, f, n and o are removed. The items retained are l, a, d, p, h and i. Using the ordered database shown in Table 2.2, the FP tree given in Figure 2.7 is constructed.

The root node points to items which are the starting items of the transactions. Here, since all transactions start with l, the root node points to l. For the first transaction, the link is made from the root node to l, from l to a, from a to d, from d to p, from p to h and from h to j. A count of 1 is stored for each of these items. The second transaction is then processed. From the root node, it moves to node l, which already exists. Its count is increased by 1. Since node d is not the next node after l, another link is made from l to d, and from d to p and from p to h. The count for l will be 2, the count for d, p and h will be 1. The next transaction moves from root node to l and from l to a and then to d and then to p and then to h along the path which already exists. The count of the items along this path is increased by 1 so that the count for l will be 3 and the count for a, d, p and h will become 2. Taking the transaction for digit 9, there is a path from root node to l, to p passing through a and d. From p, a new link is made to node i. Now the count for l will be 4 and the count for a, d and p will be 3 and the count for i will be 1. For the last transaction, the path is from root node to the already existing l and a, so that the count of l will become 5 and the

count for a will be 4. Then a new link is made from a to i, giving a count of 1 to i. As can be seen from Figure 2.7, the header node for i points to all the nodes with item i. Similarly, the header node for d, p and h also point to all the nodes having these items.

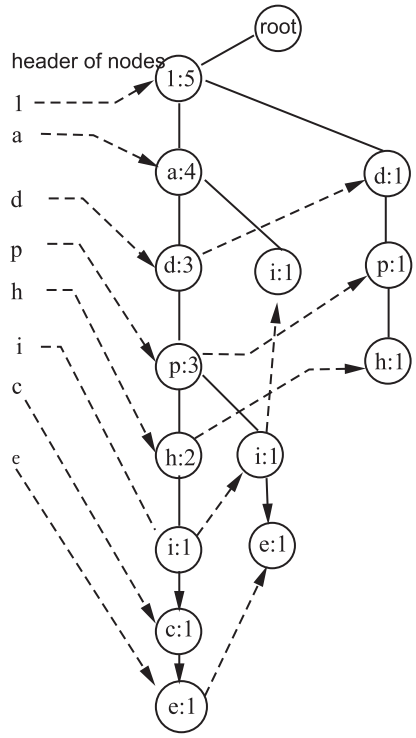


Figure 2.7 FP tree for the transaction database in Table 2.1

2.2 Representation of Clusters

Clustering refers to the process of grouping patterns with similar features together and placing objects with different features in separate groups. There are two data structures here. One is the partition P of the patterns and the other is the set of cluster representatives C.

In problems where the centroid is used as the representative of a group of patterns, P and C depend upon each other. So, if one of them is given, the other can be calculated. If the cluster centroids are given, any pattern can be optimally classified by assigning it to the cluster whose centroid is closest to the pattern. Similarly, if the partition is given, the centroids can be computed. If P is given, then C can be

computed in $O(N)$ running time if there are N patterns. If C is given, then P can be generated in $O(NM)$ time if there are M centroids.

Clusters can therefore be represented either by P or C (in the case where the centroid is used as the representative of a group of patterns) or by both P and C.

2.3 Proximity Measures

In order to classify patterns, they need to be compared against each other and against a standard. When a new pattern is present and it is necessary to classify it, the proximity of this pattern to the patterns in the training set is to be found. Even in the case of unsupervised learning, it is required to find some groups in the data so that patterns which are similar are put together. A number of similarity and dissimilarity measures can be used. For example, in the case of the nearest neighbour classifier, a training pattern which is closest to the test pattern is to be found.

2.3.1 Distance Measure

A distance measure is used to find the dissimilarity between pattern representations. Patterns which are more similar should be closer. The distance function could be a metric or a non-metric. A metric is a measure for which the following properties hold :

1. *Positive reflexivity* $d(x, x) = 0$
2. *Symmetry* $d(x, y) = d(y, x)$
3. *Triangular inequality* $d(x, y) \leq d(x, z) + d(z, y)$

The popularly used distance metric called the Minkowski metric is of the form

$$d^m(X, Y) = \left(\sum_{k=1}^{d} \mid x_k - y_k \mid^m \right)^{\frac{1}{m}}$$

When m is 1 it is called the Manhattan distance or the L_1 distance. The most popular is the Euclidean distance or the L_2 distance which occurs when m is assigned the value of 2. We then get

$$d^2(X, Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ...(x_d - y_d)^2}$$

In this way, L_∞ will be

$$d^\infty(X, Y) = \max_{k=1,..,d} \mid x_k - y_k \mid$$

While using the distance measure, it should be ensured that all the features have the same range of values, failing which attributes with larger ranges will be treated as more important. It will be like giving it a larger weightage. To ensure that all features are in the same range, normalisation of the feature values has to be carried out.

The Mahalanobis distance is also a popular distance measure used in classification. It is given by

$$d(X,Y)^2 = (X-Y)^T \Sigma^{-1} (X-Y)$$

where Σ is the covariance matrix.

EXAMPLE 4

If $X = (4, 1, 3)$ and $Y = (2, 5, 1)$ then the Euclidean distance will be

$$d(X,Y) = \sqrt{(4-2)^2 + (1-5)^2 + (3-1)^2} = 4.9$$

2.3.2 Weighted Distance Measure

When attributes need to treated as more important, a weight can be added to their values. The weighted distance metric is of the form

$$d(X,Y) = \left(\sum_{k=1}^{d} w_k \times (x_k - y_k)^m \right)^{\frac{1}{m}}$$

where w_k is the weight associated with the kth dimension (or feature).

EXAMPLE 5

If $X = (4, 2, 3)$, $Y = (2, 5, 1)$ and $w_1 = 0.3$, $w_2 = 0.6$ and $w_3 = 0.1$, then

$$d^2(X,Y) = \sqrt{0.3 \times (4-2)^2 + 0.6 \times (1-5)^2 + 0.1 \times (3-1)^2} = 3.35$$

The weights reflects the importance given to each feature. In this example, the second feature is more important than the first and the third feature is the least important.

It is possible to view the Mahalanobis distance as a weighted Euclidean distance, where the weighting is determined by the range of variability of the sample point expressed by the covariance matrix, where σ_i^2 is the variance in the ith feature direction, $i = 1, 2$. For example, if

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$$

the Mahalanobis distance gives the Euclidean distance weighted by the inverse of the variance in each dimension.

Another distance metric is the Hausdorff distance. This is used when comparing two shapes as shown in Figure 2.8. In this metric, the points sampled along the shapes boundaries are compared. The Hausdorff distance is the maximum distance between any point in one shape and the point that is closest to it in the other. If there are two point sets (I) and (J), then the Hausdorff distance is

$$\max(\max_{i\in I}\min_{j\in J} \parallel i-j \parallel, \max_{j\in J}\min_{i\in I} \parallel i-j \parallel)$$

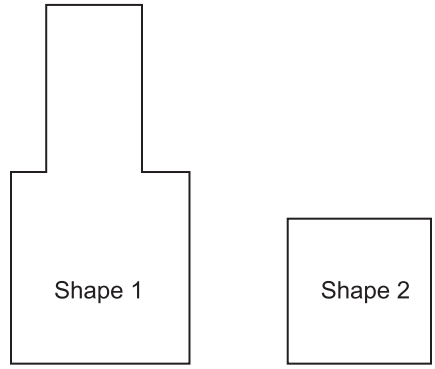


Figure 2.8 Shapes which can be compared by the Hausdorff distance

2.3.3 Non-Metric Similarity Function

Similarity functions which do not obey either the triangular inequality or symmetry come under this category. Usually these similarity functions are useful for images or string data. They are robust to outliers or to extremely noisy data. The squared Euclidean distance is itself an example of a non-metric, but it gives the same ranking as the Euclidean distance which is a metric. One non-metric similarity function is the k-median distance between two vectors. If $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$, then

$$d(X, Y) = k\text{-median}\{|x_1 - y_1|, ..., |x_n - y_n|\}$$

where the k-median operator returns the kth value of the ordered difference vector.

EXAMPLE 6

If X = (50, 3, 100, 29, 62, 140) and Y = (55, 15, 80, 50, 70, 170), then

Difference vector = {5, 12, 20, 21, 8, 30}

$$d(X, Y) = k\text{-median}\{5, 8, 12, 20, 21, 30\}$$

If $k = 3$, then $d(X, Y) = 12$

Another measure of similarity between two patterns X and Y is

$$S(X, Y) = \frac{X^t Y}{||X||\,||Y||}$$

This corresponds to the cosine of the angle between X and Y. $S(X, Y)$ is the similarity between X and Y. If we view $1 - S(X, Y)$ as the distance, $d(X, Y)$, between X and Y, then $d(X, Y)$ does not satisfy the triangular inequality, it is not a metric. However, it is symmetric, because $\cos(\theta) = \cos(-\theta)$.

EXAMPLE 7

If X, Y, and Z are two vectors in a 2-d space such that the angle between X and Y is 45 and that between Y and Z is 45, then

$$d(X, Z) = 1 - 0 = 1$$

whereas $d(X, Y) + d(Y, Z) = 2 - \sqrt{(}2) = 0.586$

So, triangular inequality is violated.

A non-metric which is non-symmetric is the Kullback–Leibler distance (KL distance). This is the natural distance function from a ''true'' probability distribution p, to a ''target'' probability distribution q. For discrete probability distributions if $p = \{p_1, ..., p_n\}$ and $q = \{q_1, ..., q_n\}$, then the KL distance is defined as

$$\text{KL}(p, q) = \Sigma_i p_i \log_2 \left(\frac{p_i}{q_i}\right)$$

For continuous probability densities, the sum is replaced by an integral.

2.3.4 Edit Distance

Edit distance measures the distance between two strings. It is also called the Levenshtein distance. The edit distance between two strings s_1 and s_2 is defined as the minimum number of point mutations required to change s_1 to s_2. A point mutation involves any one of the following operations.

1. Changing a letter
2. Inserting a letter
3. Deleting a letter

The following recurrence relation defines the edit distance between two strings

$$\begin{aligned} d(\text{“ ”}, \text{“ ”}) &= 0 \\ d(s, \text{“ ”}) &= d(\text{“ ”}, s) = \|s\| \end{aligned}$$

$$d(s_1 + ch_1, s_2 + ch_2) = \min(d(s_1, s_2) + \{\text{if } ch_1 = ch_2 \text{ then 0 else 1}\},$$

$$d(s_1 + ch_1, s_2) + 1, d(s_1, s_2 + ch_2) + 1)$$

If the last characters of the two strings ch_1 and ch_2 are identical, they can be matched for no penalty and the overall edit distance is $d(s_1, s_2)$. If ch_1 and ch_2 are different, then ch_1 can be changed into ch_2, giving an overall cost of $d(s_1, s_2) + 1$. Another possibility is to delete ch_1 and edit s_1 into $s_2 + ch_2$, i.e., $d(s_1, s_2 + ch_2) + 1$. The other possibility is $d(s_1 + ch_1, s_2) + 1$. The minimum value of these possibilities gives the edit distance.

EXAMPLE 8

1. If s = ‘‘TRAIN’’ and t = ‘‘BRAIN’’, then edit distance = 1 because using the recurrence relation defined earlier, this requires a change of just one letter.
2. If s = ‘‘TRAIN’’ and t = ‘‘CRANE’’, then edit distance = 3. We can write the edit distance from s to t to be the edit distance between ‘‘TRAI’’ and ‘‘CRAN’’ + 1 (since N and E are not the same). It would then be the edit distance between ‘‘TRA’’ and ‘‘CRA’’ + 2 (since I and N are not the same). Proceeding in this way, we get the edit distance to be 3.

2.3.5 Mutual Neighbourhood Distance (MND)

The similarity between two patterns A and B is

$$S(A, B) = f(A, B, \epsilon)$$

where ϵ is the set of neighbouring patterns. ϵ is called the *context* and corresponds to the surrounding points. With respect to each data point, all other data points are numbered from 1 to $N - 1$ in increasing order of some distance measure, such that the nearest neighbour is assigned value 1 and the farthest point is assigned the value $N - 1$. If we denote by NN(u,v), the number of data point v with respect to u, the mutual neighbourhood distance (MND), is defined as

$$\text{MND(u,v)} = \text{NN(u,v)} + \text{NN(v,u)}$$

This is symmetric and by defining NN(u,u)= 0, it is also reflexive. However, the triangle inequality is not satisfied and therefore MND is not a metric.

EXAMPLE 9

Consider Figure 2.9. In Figure 2.9(a), the ranking of the points A, B and C can be represented as

	1	2
A	B	C
B	A	C
C	B	A

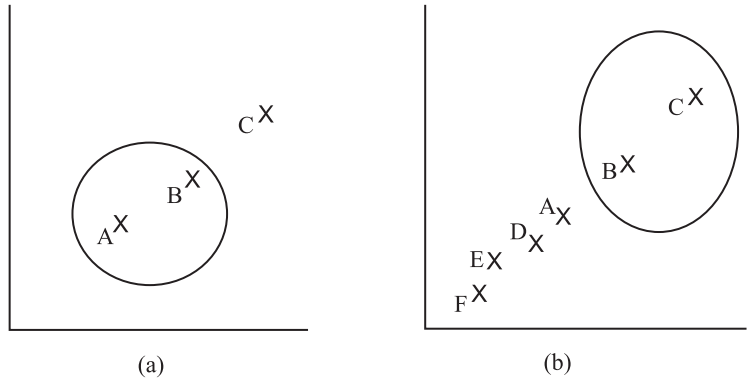
MND(A, B) = 2
MND(B, C) = 3
MND(A, C) = 4

In Figure 2.9(b), the ranking of the points A, B, C, D, E and F can be represented as

	1	2	3	4	5
A	D	E	F	B	C
B	A	C	D	E	F
C	B	A	D	E	F

MND(A, B) = 5
MND(B, C) = 3
MND(A, C) = 7

It can be seen that in the first case, the least MND distance is between A and B, whereas in the second case, it is between B and C. This happens by changing the context.



(a) (b)

Figure 2.9 Mutual neighbourhood distance

2.3.6 Conceptual Cohesiveness

In this case, distance is applied to pairs of objects based on a set of concepts. A concept is a description of a class of objects sharing some attribute value. For example, (colour = blue) is a concept that represents a collection of blue objects. Conceptual cohesiveness uses the domain knowledge in the form of a set of concepts to characterise the similarity. The conceptual cohesiveness (similarity function) between A and B is characterised by

$S(A, B) = f(A, B, \epsilon, \mathcal{C})$, where $\mathcal{C}$ is a set of pre-defined concepts.

The notion of conceptual distance combines both symbolic and numerical methods. To find the conceptual distance between patterns A and B, A and B are generalised and the similar and dissimilar predicates will give the similarity $S(A, B, G)$ and $D(A, B, G)$. This depends on the generalisation $G(A, B)$. The distance function $f(A, B, G)$ is given by

$$f(A, B, G) = \frac{D(A, B, G)}{S(A, B, G)}$$

The generalisation is not unique—there may be other generalisations. So we can get $S(A, B, G')$ and $D(A, B, G')$ for another generalisation $G'(A, B)$, giving the distance function

$$f(A, B, G') = \frac{D(A, B, G')}{S(A, B, G')}$$

The minimum of these distance functions gives the conceptual distance. The reciprocal of the conceptual distance is called the *conceptual cohesiveness*.

2.3.7 Kernel Functions

The kernel function can be used to characterise the distance between two patterns x and y.

1. *Polynomial kernel* The similarity between x and y can be represented using the polynomial kernel function

   $$K(x, y) = \phi(x)^t \phi(y) = (x^t y + 1)^2$$

   where $\phi(x) = (x_1^2, x_2^2, 1, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2)$

   By using this, linearly dependent vectors in the input space get transformed to independent vectors in kernel space.

2. *Radial basis function kernel* Using the RBF kernel,

$$K(x, y) = \exp^{\frac{-||x-y||^2}{2\sigma^2}}$$

The output of this kernel depends on the Euclidean distance between x and y. Either x or y will be the centre of the radial basis function and σ will determine the area of influence over the data space.

2.4 Size of Patterns

The size of a pattern depends on the attributes being considered. In some cases, the length of the patterns may be a variable. For example, in document retrieval, documents may be of different lengths. This can be handled in different ways.

2.4.1 Normalisation of Data

This process entails normalisation so that all patterns have the same dimensionality. For example, in the case of document retrieval, a fixed number of keywords can be used to represent the document. Normalisation can also be done to give the same importance to every feature in a data set of patterns.

EXAMPLE 10

Consider a data set of patterns with two features as shown below :

X_1 : (2, 120)
X_2 : (8, 533)
X_3 : (1, 987)
X_4 : (15, 1121)
X_5 : (18, 1023)

Here, each line corresponds to a pattern. The first value represents the first feature and the second value represents the second feature. The first feature has values below 18, whereas the second feature is much larger. If these values are used in this way for computing distances, the first feature will be insignificant and will not have any bearing on the classification. Normalisation gives equal importance to every feature. Dividing every value of the first feature by its maximum value, which is 18, and dividing every value of the second feature by its maximum value, which is 1121, will make all the values lie between 0 and 1 as shown below :

$X_1^{'}$: (0.11, 0.11)
$X_2^{'}$: (0.44, 0.48)

$X_3^{'}$: (0.06, 0.88)
$X_4^{'}$: (0.83, 1.0)
$X_5^{'}$: (1.0, 0.91)

2.4.2 Use of Appropriate Similarity Measures

Similarity measures can deal with unequal lengths. One such similarity measure is the edit distance.

2.5 Abstractions of the Data Set

In supervised learning, a set of training patterns where the class label for each pattern is given, is used for classification. The complete training set may not be used because the processing time may be too long but an abstraction of the training set can be used. For example, when using the nearest neighbour classifier on a test pattern with n training patterns, the effort involved will be $O(n)$. If m patterns are used and $(m << n)$, $O(m)$ effort will be sufficient. Depending on the abstraction, different classifiers are used.

1. *No abstraction of patterns* All the training patterns are used and abstraction of the data is not carried out. Examples of classifiers used here are the nearest neighbour (NN), the k-nearest neighbour (kNN) and the modified kNN (MkNN) classifiers. These methods use the neighbour(s) of the test pattern from all the training patterns.
2. *Single representative per class* All the patterns belonging to a class are represented by a single representative pattern. This single representative can be obtained in many ways. One option is to take the mean of all the patterns in the class. One could also take the medoid of all the samples. Here by medoid, we mean the most centrally located pattern. Other methods of obtaining a single representative, which may be typical of the domain to which the patterns belong, can also be used. A test pattern would be compared to the centroid of the patterns belonging to each class and classified as belonging to the class of the mean closest to it. This is the minimum distance classifier (MDC) .
3. *Multiple representatives per class*
   (a) *Cluster representatives as abstractions* The training patterns in the class are clustered and the cluster centre of each cluster is the representative of

all the patterns in the cluster. The set of cluster centres is an abstraction of the whole training set.

(b) *Support vectors as representatives* Support vectors are determined for the class and used to represent the class. Support vector machines (SVMs) are described in Chapter 7.

(c) *Frequent item set based abstraction* In the case of transaction data bases, each pattern represents a transaction. An example of this is the departmental store where each transaction is the set of items bought by one customer. These transactions or patterns are of variable length. Item sets which occur frequently are called frequent item sets. If we have a threshold α, item sets which occur more than α times in the data set are the frequent item sets. Frequent item sets are an abstraction of the transaction database. An important observation is that any discrete-valued pattern may be viewed as a transaction.

EXAMPLE 11

In Figure 2.10, the cluster of points can be represented by its centroid or its medoid. Centroid stands for the sample mean of the points in the cluster. It need not coincide with one of the points in the cluster. The medoid is the most centrally located point in the cluster. Use of the centroid or medoid to represent the cluster is an example of using a single representative per class. It is possible to have more than one representative per cluster. For example, four extreme

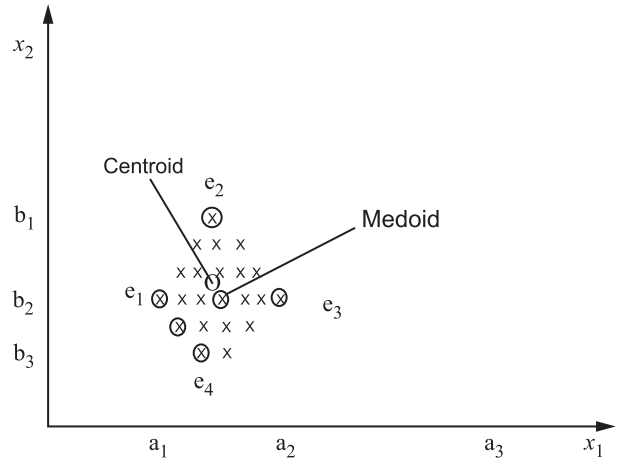


Figure 2.10 A set of data points

points labelled e_1, e_2, e_3, e_4 can represent the cluster as shown in Figure 2.10. When one representative point is used to represent the cluster, in the case of the nearest neighbour classification, only one distance needs to be computed from a test point instead of, say in our example, 22 distances. In the case of there being more than one representative pattern, if four representative patterns are there, only four distances need to be computed instead of 22 distances.

2.6 Feature Extraction

Feature extraction involves detecting and isolating various desired features of patterns. It is the operation of extracting features for identifying or interpreting meaningful information from the data. This is especially relevant in the case of image data where feature extraction involves automatic recognition of various features. Feature extraction is an essential pre-processing step in pattern recognition.

2.6.1 Fisher's Linear Discriminant

Fisher's linear discriminant projects high-dimensional data onto a line and performs classification in this space. If there are two classes, the projection maximises the distance between the means and minimises the variance within each class. Fisher's criterion which is maximised over all linear projections V can be defined as :

$$J(\mathrm{V}) = \frac{|\ \mathrm{mean}_1 - \mathrm{mean}_2|^2}{s_1^2 + s_2^2}$$

where mean_1 and mean_2 represent the mean of Class 1 patterns and Class 2 patterns respectively and s^2 is proportional to the variance. Maximising this criterion yields a closed form solution that involves the inverse of a covariance matrix.

In general, if x_i is a set of N column vectors of dimension D, the mean of the data set is

$$\mathrm{mean} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

In case of multi-dimensional data, the mean is a vector of length D, where D is the dimension of the data.

If there are K classes $\{C_1, C_2, ..., C_K\}$, the mean of class C_k containing N_k members is

$$\text{mean}_k = \frac{1}{N_k} \sum_{x_i \in C_k} x_i$$

The between class scatter matrix is

$$\sigma_B = \sum_{k=1}^{K} N_k (\text{mean}_k - \text{mean})(\text{mean}_k - \text{mean})^T$$

The within class scatter matrix is

$$\sigma_W = \sum_{k=1}^{K} \sum_{x_i \in C_k} (x_i - \text{mean}_k)(x_i - \text{mean}_k)^T$$

The transformation matrix that re-positions the data to be most separable is

$$J(V) = \frac{V^T \sigma_B V}{V^T \sigma_W V}$$

$J(V)$ is the criterion function to be maximised. The vector V that maximises $J(V)$ can be shown to satisfy

$$\sigma_B V = \lambda \sigma_W V$$

Let $\{v_1, v_2, ..., v_D\}$ be the generalised eigenvectors of σ_B and σ_W.

This gives a projection space of dimension D. A projection space of dimension $d < D$ can be defined using the generalised eigenvectors with the largest d eigenvalues to give $V_d = [v_1, v_2, ..., v_d]$.

The projection of vector x_i into a sub-space of dimension d is $y = V_d^T x$. In the case of the two-class problem,

$$\text{mean}_1 = \frac{1}{N_1} \sum_{x_i \in C_1} x_i$$

$$\text{mean}_2 = \frac{1}{N_2} \sum_{x_i \in C_2} x_i$$

$$\sigma_B = N_1 (\text{mean}_1 - \text{mean})(\text{mean}_1 - \text{mean})^T + N_2 (\text{mean}_2 - \text{mean})(\text{mean}_2 - \text{mean})^T$$

$$\sigma_W = \sum_{x_i \in C_1} (x_i - \text{mean}_1)(x_i - \text{mean}_1)^T + \sum_{x_i \in C_2} (x_i - \text{mean}_2)(x_i - \text{mean}_2)^T$$

$$\sigma_B V = \lambda \sigma_W V$$

This means that

$$\sigma_W^{-1}\sigma_B V = \lambda V$$

Since $\sigma_B V$ is always in the direction of $\text{mean}_1 - \text{mean}_2$, the solution for V is :

$$V = \sigma_W^{-1}(\text{mean}_1 - \text{mean}_2)$$

The intention here is to convert a d-dimensional problem to a one-dimensional one.

EXAMPLE 12

If there are six points namely $(2,2)^t$, $(4,3)^t$ and $(5,1)^t$ belonging to Class 1 and $(1,3)^t$, $(5,5)^t$ and $(3,6)^t$ belonging to Class 2, the means will be

$$m_{x1} = \begin{bmatrix} 3.66 \\ 2 \end{bmatrix}$$

$$m_{x2} = \begin{bmatrix} 3 \\ 4.66 \end{bmatrix}$$

The within class scatter matrix will be

$$\sigma_W = \begin{bmatrix} -1.66 \\ 0 \end{bmatrix} \times \begin{bmatrix} -1.66 & 0 \end{bmatrix} + \begin{bmatrix} 0.33 \\ 1 \end{bmatrix} \times \begin{bmatrix} 0.33 & -1 \end{bmatrix} + \begin{bmatrix} 1.33 \\ -1 \end{bmatrix}$$
$$\times \begin{bmatrix} 1.33 & -1 \end{bmatrix} + \begin{bmatrix} -2 \\ -1.66 \end{bmatrix} \times \begin{bmatrix} -2 & -1.66 \end{bmatrix} + \begin{bmatrix} 2 \\ 0.33 \end{bmatrix}$$
$$\times \begin{bmatrix} 2 & 0.33 \end{bmatrix} + \begin{bmatrix} 0 \\ 1.33 \end{bmatrix} \times \begin{bmatrix} 0 & 1.33 \end{bmatrix} = \begin{bmatrix} 12.63 & 2.98 \\ 2.98 & 6.63 \end{bmatrix}$$

$$S_W^{-1} = \frac{1}{74.88} \begin{bmatrix} 6.63 & -2.98 \\ -2.98 & 12.63 \end{bmatrix}$$

The direction is given by

$$V = \sigma_W^{-1}(\text{mean}_1 - \text{mean}_2) = \frac{1}{74.88} \begin{bmatrix} 6.63 & -2.98 \\ -2.98 & 12.63 \end{bmatrix} \times \begin{bmatrix} 0.66 \\ -2.66 \end{bmatrix}$$

$$V = \frac{1}{74.88} \begin{bmatrix} 12.30 \\ -34.2 \end{bmatrix} = \begin{bmatrix} 0.164 \\ -0.457 \end{bmatrix}$$

Note that $v^t x \geq -0.586$ if $x \in$ Class 1 and $v^t x \leq -1.207$ if $x \in$ Class 2.

2.6.2 Principal Component Analysis (PCA)

PCA involves a mathematical procedure that transforms a number of correlated variables into a smaller number of uncorrelated variables called principal components. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible. PCA finds the most accurate data representation in a lower dimensional space. The data is projected in the direction of maximum variance.

If x is a set of N column vectors of dimension D, the mean of the data set is

$$m_x = E\{x\}$$

The covariance matrix is

$$C_x = E\{(x - m_x)(x - m_x)^T\}$$

The components of C_x denoted by c_{ij} represent the covariances between the random variable components x_i and x_j. The component c_{ii} is the variance of the component x_i.

This is a symmetric matrix from which the orthogonal basis can be calculated by finding its eigenvalues and eigenvectors. The eigenvectors e_i and the corresponding eigenvalues λ_i are solutions of the equation

$$C_x e_i = \lambda_i e_i, i = 1, ..., n$$

By ordering the eigenvectors in the order of descending eigenvalues, an ordered orthogonal basis can be created with the first eigenvector having the direction of the largest variance of the data. In this way, we can find directions in which the data set has the most significant amounts of energy.

If A is the matrix consisting of eigenvectors of the covariance matrix as the row vectors formed by transforming a data vector x, we get

$$y = A(x - m_x)$$

The original data vector x can be reconstructed from y by

$$x = A^T y + m_x$$

Instead of using all the eigenvectors, we can represent the data in terms of only a few basis vectors of the orthogonal basis. If we denote the matrix having the K first eigenvectors as A_K, we get

$$y = A_K(x - m_x)$$

and

$$x = A_K^T y + m_x$$

The original data vector is projected on the coordinate axes having the dimension K and the vector is transformed back by a linear combination of the basis vectors. This minimises the mean-square error with the given number of eigenvectors used. By picking the eigenvectors having the largest eigenvalues, as little information as possible is lost. This provides a way of simplifying the representation by reducing the dimension of the representation.

EXAMPLE 13

A data set contains four patterns in two dimensions. The patterns belonging to Class 1 are (1, 1) and (1, 2). The patterns belonging to Class 2 are (4, 4) and (5, 4). With these four patterns, the covariance matrix is

$$C_x = \begin{bmatrix} 4.25 & 2.92 \\ 2.92 & 2.25 \end{bmatrix}$$

The eigenvalues of C_x are

$$\lambda = \begin{bmatrix} 6.336 \\ 0.1635 \end{bmatrix}$$

where the two eigenvalues are represented as a column vector.

Since the second eigenvalue is very small compared to the first eigenvalue, the second eigenvector can be left out. The eigenvector which is most significant is

$$e_1 = \begin{bmatrix} 0.814 \\ 0.581 \end{bmatrix}$$

To transform the patterns on to the eigenvector, the pattern (1, 1) gets transformed to

$$\begin{bmatrix} 0.814 & 0.581 \end{bmatrix} \times \begin{bmatrix} -1.75 \\ -1.75 \end{bmatrix} = -2.44$$

Similarly, the patterns (1, 2), (4, 4) and (5, 4) get transformed to -1.86, 1.74 and 2.56.

When we try to get the original data using the transformed data, some information is lost. For pattern (1, 1), using the transformed data, we get

$$\begin{bmatrix} 0.814 & 0.581 \end{bmatrix}^T \times (-2.44) + \begin{bmatrix} 2.75 \\ 2.75 \end{bmatrix} = \begin{bmatrix} -1.99 & -1.418 \end{bmatrix}^T + \begin{bmatrix} 2.75 \\ 2.75 \end{bmatrix}$$

$$= \begin{bmatrix} 0.76 \\ 1.332 \end{bmatrix}$$

2.7 Feature Selection

The features used for classification may not always be meaningful. Removal of some of the features may give a better classification accuracy. Features which are useless for classification are found and left out. Feature selection can also be used to speed up the process of classification, at the same time, ensuring that the classification accuracy is optimal. Feature selection ensures the following:

1. *Reduction in cost of pattern classification and design of the classifier* Dimensionality reduction, i.e., using a limited feature set simplifies both the representation of patterns and the complexity of the classifiers. Consequently, the resulting classifier will be faster and use less memory.
2. *Improvement of classification accuracy* The improvement in classification is due to the following reasons:

   (a) The performance of a classifier depends on the inter-relationship between sample sizes, number of features, and classifier complexity. To obtain good classification accuracy, the number of training samples must increase as the number of features increase. The probability of misclassification does not increase as the number of features increases, as long as the number of training patterns is arbitrarily large. Beyond a certain point, inclusion of additional features leads to high probabilities of error due to the finite number of training samples. If the number of training patterns used to train the classifier is small, adding features may actually degrade the performance of a classifier. This is called the peaking phenomena and is illustrated in Figure 2.11. Thus it can be seen that a small number of features can alleviate the curse of dimensionality when the number of training patterns is limited.

   (b) It is found that under a broad set of conditions, as dimensionality increases, the distance to the nearest data point approaches the distance to the furthest data point. This affects the results of the nearest neighbour problem. Reducing the dimensionality is meaningful in such cases to get better results.

All feature selection algorithms basically involve searching through different feature sub-sets. Since feature selection algorithms are basically search procedures, if the number of features is large (or even above, say, 30 features), the number of feature sub-sets become prohibitively large. For optimal algorithms such as the exhaustive search and branch and bound technique , the computational efficiency comes down rather steeply and it is necessary to use sub-optimal procedures which are essentially faster.

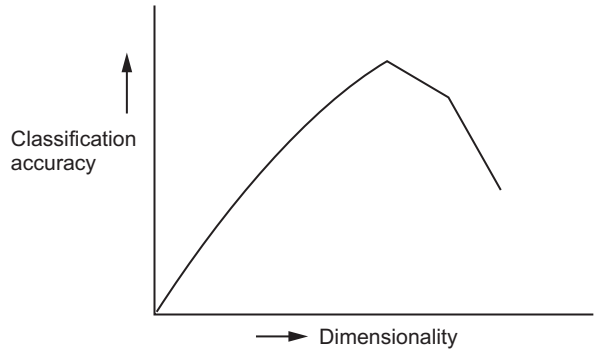


Figure 2.11 Peaking phenomenon or the curse of dimensionality

Feature sub-sets that are newly discovered have to be evaluated by using a criterion function. For a feature sub-set X, we have to find $J(X)$. The criterion function usually used is the classification error on the training set. Here $J = (1 - P_e)$, where $P(e)$ is the probability of classification error. This suggests that a higher value of J gives a better feature sub-set.

2.7.1 Exhaustive Search

The most straightforward approach to the problem of feature selection is to search through all the feature sub-sets and find the best sub-set. If the patterns consist of d features, and a sub-set of size m features is to be found with the smallest classification error, it entails searching all $\binom{d}{m}$ possible sub-sets of size m and selecting the sub-set with the highest criterion function $J(.)$, where $J = (1 - P_e)$. Table 2.3 shows the various sub-sets for a data set with 5 features. This gives sub-sets of three features. A 0 means that the corresponding feature is left out and a 1 means that the feature is included.

Table 2.3 Features selected in exhaustive search

Sl. No.	f1	f2	f3	f4	f5
1.	0	0	1	1	1
2.	0	1	0	1	1
3.	0	1	1	0	1
4.	0	1	1	1	0
5.	1	0	0	1	1
6.	1	0	1	0	1
7.	1	0	1	1	0
8.	1	1	0	0	1
9.	1	1	0	1	0
10.	1	1	1	0	0

This technique is impractical to use even for moderate values of d and m. Even when d is 24 and m is 12, approximately 2.7 million feature sub-sets must be evaluated.

2.7.2 Branch and Bound Search

The branch and bound scheme avoids exhaustive search by using intermediate results for obtaining bounds on the final criterion value. This search method assumes monotonicity as described below.

Let $(Z_1, Z_2, ..., Z_l)$ be the $l = d - m$ features to be discarded to obtain an m feature sub-set. Each variable Z_i can take on values in {1, 2,..., d}. The order of the Z_is is immaterial and they are distinct, so we consider only sequences of Z_i such that $Z_1 < Z_2 < ... < Z_l$. The feature selection criterion is $J_l(Z_1, \ldots, Z_l)$. The feature sub-set selection problem is to find the optimum sub-set $Z_1^*, \ldots, Z_l^*$ such that

$$J_l(Z_1^*, ..., Z_l^*) = \max J_l(Z_1, ..., Z_l)$$

If the criterion J satisfies monotonicity, then

$$J_1(Z_1) \geq J_2(Z_1, Z_2) \geq ... \geq J_l(Z_1, ..., Z_l)$$

This means that a sub-set of features should not be better than any larger set that contains the sub-set.

Let B be a lower bound on the optimum value of the criterion $J_l(Z_1^*, ..., Z_l^*)$. That is

$$B \leq J_l(Z_1^*, ..., Z_l^*)$$

If $J_k(Z_1, ..., Z_k)(k \leq l)$ were less than B, then

$$J_l(Z_1, ..., Z_k, Z_{k+1}, ..., Z_l) \leq B$$

for all possible $Z_{k+1}, ..., Z_l$.

This means that whenever the criterion evaluated for any node is less than the bound B, all nodes that are successors of that node also have criterion values less than B, and therefore cannot lead to the optimum solution. This principle is used in the branch and bound algorithm. The branch and bound algorithm successively generates portions of the solution tree and computes the criterion. Whenever a sub-optimal partial sequence or node is found to have a criterion lower than the bound at that point in time, the sub-tree under that node is implicitly rejected and other partial sequences are explored.

Starting from the root of the tree, the successors of the current node are enumerated

in the ordered list LIST(i). The successor for which the partial criterion $J_i(Z_1, ..., Z_i)$ is maximum is picked as the new current node and the algorithm moves on to the next higher level. The lists LIST(i) at each level i keeps track of the nodes that have been explored. The SUCCESSOR variables determine the number of successor nodes the current node will have at the next level. AVAIL keeps track of the feature values that can be enumerated at any level. Whenever the partial criterion is found to be less than the bound, the algorithm backtracks to the previous level and selects a hitherto unexplored node for expansion. When the algorithm reaches the last level l, the lower bound B is updated to be the current value of $J_l(Z_1, ..., Z_l)$ and the current sequence $(Z_1, ..., Z_l)$ is saved as $(Z_1^*, ..., Z_l^*)$. When all the nodes in LIST(i) for a given i are exhausted, the algorithm backtracks to the previous level. When the algorithm backtracks to level 0, it terminates. At the conclusion of the algorithm, $(Z_1^*, ..., Z_l^*)$ will give the complement of the best feature set.

The branch and bound algorithm is as follows:

STEP 1: Take root node as the current node.

STEP 2: Find successors of the current node.

STEP 3: If successors exist, select the successor i not already searched which has the maximum J_i and make it the current node.

STEP 4: If it is the last level, update the bound B to the current value of $J_l(Z_1, ..., Z_l)$ and store the current path $(Z_1, ..., Z_l)$ as the best path $(Z_1^*, ..., Z_l^*)$. Backtrack to previous levels to find the current node.

STEP 5: If the current node is not the root, go to Step 2.

STEP 6: The complement of the best path at this point $(Z_1^*, ..., Z_l^*)$ gives the best feature set.

This algorithm assumes monotonicity of the criterion function $J(.)$. This means that for two feature sub-sets χ_1 and χ_2, where $\chi_1 \subset \chi_2$, $J(\chi_1) < J(\chi_2)$. This is not always true.

The modified branch and bound algorithm (BAB$^+$) gives an algorithmic improvement on the BAB. Whenever the criterion evaluated for any node is not larger than the bound B, all nodes that are its successors also have criterion values not larger than B and therefore cannot be the optimal solution. BAB$^+$ does not generate these nodes and replaces the current bound with the criterion value which is larger than it and is held by the terminal node in the search procedure. The bound reflecting the criterion value held by the globally optimal solution node will never be replaced. This algorithm implicitly skips over the intermediate nodes and ''short-traverses'', thus improving the efficiency of the algorithm.

The relaxed branch and bound (RBAB) can be used even if there is a violation of monotonicity. Here we have a margin b and even if J exceeds the bound by an amount less than b, the search is continued. b is called the margin. On the other hand,

the modified relaxed branch and bound algorithm gets rid of the margin and cuts branches below a node Z only if both Z and a parent of Z have a criterion value less than the bound.

EXAMPLE 14

Consider a data set having four features f1, f2, f3 and f4. Figure 2.12 shows the tree that results when one feature at a time is removed. The numbering of the nodes shows the order in which the tree is formed. When a leaf node is reached, the criterion value of the node is evaluated. When node 3 is reached, since its criterion value is 25, the bound is set as 25. Since the bound of the next node 4 is 45 which is greater than 25, the bound is revised to 45. Node 5 has a smaller criterion than the bound and therefore the bound remains as 45. When node 6 is evaluated, it is found to have a criterion of 37. Since this is smaller than the bound, this node is not expanded further. Since monotonicity is assumed, nodes 7 and 8 are estimated to have criteria which are smaller than 37. Similarly node 9 which has a criterion of 43 is also not expanded further since its criterion is less than the bound. The features removed would therefore be f1 and f3. This is shown in Figure 2.13. Using relaxed branch and bound, if the margin b is 5, considering node 9, since the difference between its criterion 43 and the bound which is 45 is less than the margin, node 9 is further expanded to give node 10. If monotonicity is violated and node 10 has a criterion greater than 45, it will be chosen as the one with the best criterion which makes the two features to be removed as f3 and f4. This is shown in Figure 2.13 where the criterion value is 47.

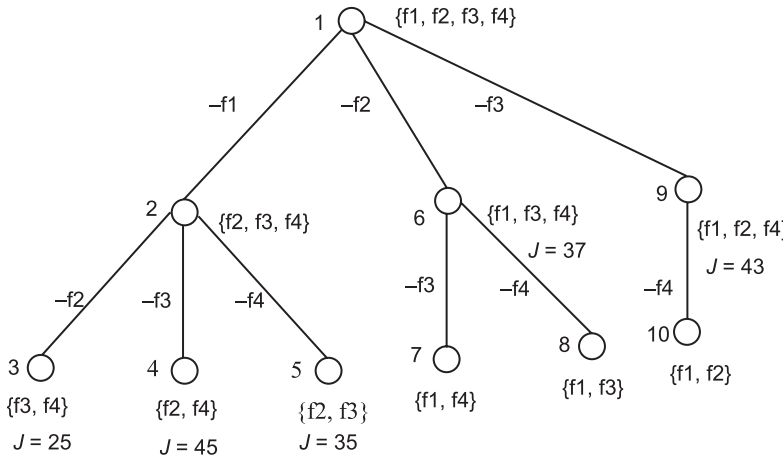


Figure 2.12 The solution tree for d = 4 and m = 2

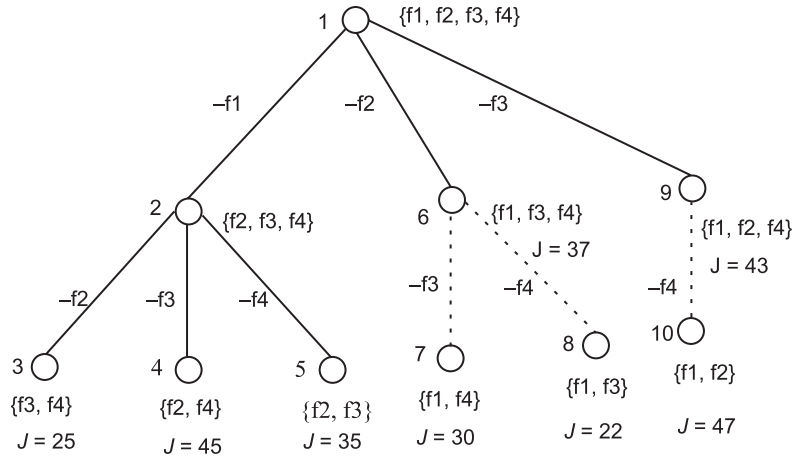


Figure 2.13 Using branch and bound algorithm

2.7.3 Selection of Best Individual Features

This is a very simple method where only the best features are selected. All the individual features are evaluated independently. The best m features are then selected. This method, though computationally simple, is very likely to fail since the dependencies between features also have to be taken into account.

2.7.4 Sequential Selection

These methods operate either by evaluating growing feature sets or shrinking feature sets. They either start with the empty set and add features one after the other; or they start with the full set and delete features. Depending on whether we start with an empty set or a full one, we have the sequential forward selection (SFS) and the sequential backward selection (SBS) respectively. Since these methods do not examine all possible sub-sets, the algorithms do not guarantee the optimal result (unlike the branch and bound and the exhaustive search techniques). Besides, these methods suffer from the ''nesting effect'' explained below.

Sequential Forward Selection (SFS)

The sequential method adds one feature at a time, each time checking on the performance. It is also called the ''bottom-up approach'' since the search starts with an empty set and successively builds up the feature sub-set. The method suffers from the ''nesting effect'' since features once selected cannot be later discarded. The number of feature sub-sets searched will be

$$\sum_{i=1}^{m}(d-i+1) = m[d - \frac{(m-1)}{2}]$$

The features are selected according to their significance. The most significant feature is selected to be added to the feature sub-set at every stage. The most significant feature is the feature that gives the highest increase in criterion function value as compared to that of the others before adding the feature. If U_0 is the feature added to the set X_k consisting of k features then the significance of this is

$$S_{k+0}(U_0) = J(X_k \bigcup U_0) - J(X_k)$$

The most significant feature with respect to the set X_k is

$$S_{k+0}(U_0^r) = \max_{1 \le i \le \phi} S_{k+0}(U_0^i)$$

i.e., $$J(X_k \bigcup U_0^r) = \max_{1 \le i \le \phi} J(X_k \bigcup U_0^i)$$

where ϕ is the number of all the possible 0-tuples.

The least significant feature with respect to set X_k is

$$S_{k+0}(U_0^r) = \min_{1 \le i \le \phi} S_{k+0}(U_0^i)$$

i.e., $$J(X_k \bigcup U_0^r) = \min_{1 \le i \le \phi} J(X_k \bigcup U_0^i)$$

Sequential Backward Selection (SBS)

This method first uses all the features and then deletes one feature at a time. It is also called the ''top-down'' approach, since the search starts with the complete set of features and successively discards features. The disadvantage is that features, once discarded, cannot be re-selected. Features are discarded successively by finding the least significant feature at that stage.

2.7.5 Sequential Floating Search

To take care of the ''nesting effect'' of SFS and SBS, ''plus l take away r'' selection was introduced, where the feature sub-set is first enhanced by l features using forward selection and then r features are deleted using backward selection. The main drawback of this method is that there is no theoretical way of predicting the values of l and r to achieve the best feature sub-set. The floating search method is an improvement on this, since there is no need to specify the parameters l and r. The

number of forward and backward steps is determined dynamically while the method is running so as to maximise the criterion function. At each step, only a single feature is added or removed. The values of l and r is kept ''floating'', i.e., they are kept flexible so as to approximate the optimal solution as much as possible. Consequently, the dimensionality of the feature sub-set does not change monotonically but is actually ''floating'' up and down.

Sequential Floating Forward Search (SFFS)

The principle of SFFS is as follows:

STEP 1: Let $k = 0$.

STEP 2: Add the most significant feature to the current sub-set of size k. Let $k = k+1$.

STEP 3: Conditionally remove the least significant feature from the current sub-set.

STEP 4: If the current sub-set is the best sub-set of size $k - 1$ found so far, let $k = k - 1$ and go to Step 3. Else return the conditionally removed feature and go to Step 2.

EXAMPLE 15

Consider a data set of patterns with four features f1, f2, f3 and f4. The steps of the algorithm are as given below.

STEP 1: Let the sub-set of the features chosen be empty, i.e., $F = \phi$.

STEP 2: The most significant feature is found. Let it be f3. Then F = {f3}.

STEP 3: The least significant feature is found, i.e., it is seen if f3 can be removed.

STEP 4: The removal of f3 does not improve the criterion value and f3 is restored.

STEP 5: The most significant feature is found. Let it be f2. Then F = {f3, f2}

STEP 6: The least significant feature is found. It is f2. Condition fails.

STEP 7: The most significant feature is found. Let it be f1. Then F = {f3, f2, f1}

STEP 8: The least significant feature is found. Let it be f3. If it gives a better sub-set, then F = {f1, f2}.

The optimal set would be F = {f1, f2} if $m = 2$. It is noted that if in the last step, the least significant feature was found to be f1, F = {f2, f3} as in Step 5. This leads to looping.

Sequential Floating Backward Search (SBFS)

This method is similar to the SFFS except that backward search is carried out first and then the forward search. The principle of SBFS is as follows:

STEP 1: $k = 0$.

STEP 2: Remove the least significant feature from the current sub-set of size k. Let $k = k - 1$.

STEP 3: Conditionally add the most significant feature from the features not in the current sub-set.

STEP 4: If the current sub-set is the best sub-set of size $k - 1$ found so far, let $k = k + 1$ and go to Step 3. Else remove the conditionally added feature and go to Step 2.

The algorithm starts with all the features and then removes them one by one. Due to this, the method does not always give good results. If the dimensionality is large, i.e., d is large and if m is very small, it will be time-consuming and it is better to use the SFFS.

2.7.6 Max–Min Approach to Feature Selection

This method has a computational advantage over the other well-known methods due to the fact that instead of computationally time-consuming calculations in a multi-dimensional space, the max–min method requires calculations in two-dimensional space only. However, the results achieved by this method are rather unsatisfactory. It works as follows.

Let us denote

f_i = feature from the selected feature set $F_k = \{f_1, ..., f_k\}$ acquired in the *i*th step of the selection procedure.

g_j = *j*th feature from the set of features not selected.

$\delta J(y_j, x_i)$ = the absolute value of the difference between $J(g_j, f_i)$ and $J(f_i)$.

In the max–min method, the new feature y_j is chosen as the next feature if it yields

$$\max_{\forall y_j} \min_{\forall x_i} \Delta J(y_j, x_i)$$

The poor results of this method confirm that it is not possible to select a set of features in a high-dimensional space based on two-dimensional information without a substantial information loss.

EXAMPLE 16

Consider a data set with patterns having four features f1, f2, f3 and f4. The criterion function value using up to two features at a time is shown in Table 2.4.

Table 2.4 Criterion function using a sub-set of features

Feature	f1	f2	f3	f4
f1	10	30	35	55
f2	30	20	40	53
f3	35	40	30	42
f4	55	53	42	40

If J(f1) represents the criterion function using only feature f1, then

$$J\ (\text{f1}) = 10;\ J\ (\text{f2}) = 20;\ J\ (\text{f3}) = 30;\ J\ (\text{f4}) = 40$$

This is depicted by the diagonal entries in Table 2.4.

First considering f1, if $J\ (\text{f1, f2}) = 30$, J(f1, f3) = 35 and $J\ (\text{f1, f4}) = 55$, then

$$\Delta J\ (\text{f1, f2}) = 10;\ \Delta J\ (\text{f1, f3}) = 5;\ \Delta J\ (\text{f1, f4}) = 15$$

The minimum of these is $\Delta J\ (\text{f1, f3}) = 5$

Next considering f2, if $J\ (\text{f2, f1}) = 30$, $J\ (\text{f2, f3}) = 40$ and $J\ (\text{f2,f4}) = 53$, then

$$\Delta J\ (\text{f2, f1}) = 20;\ \Delta J\ (\text{f2, f3}) = 10;\ \Delta J\ (\text{f2, f4}) = 13$$

The minimum of these is $\Delta J\ (\text{f2, f3}) = 10$

Next considering f3, if $J\ (\text{f3, f1}) = 35$, $J\ (\text{f3, f2}) = 40$ and $J\ (\text{f3, f4}) = 42$, then

$$\Delta J\ (\text{f3, f1}) = 25;\ \Delta J\ (\text{f3, f2}) = 20;\ \Delta J\ (\text{f3, f4}) = 2$$

The minimum of these is $\Delta J\ (\text{f3, f4}) = 2$

Next considering f4, if $J\ (\text{f4, f1}) = 55$, $J\ (\text{f4, f2}) = 53$ and $J\ (\text{f4, f3}) = 42$, then

$$\Delta J\ (\text{f4, f1}) = 45;\ \Delta J\ (\text{f4, f2}) = 33;\ \Delta J\ (\text{f4, f3}) = 12$$

The minimum of these is $\Delta J\ (\text{f4, f3}) = 12$

Finding the maximum of the four minimum values, we get f4 which is chosen as the next feature.

It can be seen that only two features are considered at a time. The features chosen could have been different if more number of features are considered at a time.

2.7.7 Stochastic Search Techniques

Genetic algorithm is a stochastic search technique which is often used for feature selection. The population in the GA consists of strings which are binary in nature. Each string (or chromosome) is of length d, with each position i being zero or one depending on the absence or presence of feature i in the set. This means that each feature sub-set is coded as a d-element bit string or binary valued vector. Each string in the population is a feature selection vector α, where each $\alpha = \alpha_1, ..., \alpha_d$ and α_i assumes a value 0 if the ith feature is excluded and 1 if it is present in the sub-set. To compute its fitness, each chromosome is evaluated by determining its performance on the training set.

2.7.8 Artificial Neural Networks

A multilayer feed-forward network with a back-propagation learning algorithm is used in this method. The approach considered here is to take a larger than necessary network and then remove unnecessary nodes. Pruning is carried out by eliminating the least salient nodes. It is based on the idea of iteratively eliminating units and adjusting the remaining weights in such a way that the network performance does not become worse over the entire training set. The pruning problem is formulated in terms of solving a system of linear equations using the optimisation technique.

The pruning of nodes corresponds to removing the corresponding features from the feature set. The saliency of a node is defined as the sum of the increase in error over all the training patterns caused by the removal of that node. The node pruning based feature selection first trains a network and then removes the least salient node. The reduced network is trained again, followed by the removal of the least salient node. This procedure is repeated to get the least classification error.

2.8 Evaluation of Classifiers

Before using a classifier to carry out a classification task, it is necessary to evaluate its performance. The various parameters of the classifier which requires to be taken into account are

1. *Accuracy of the classifier* The main aim of using a classifier is to correctly classify unknown patterns. Classification accuracy is an important parameter in evaluating a classifier.

2. *Design time and classification time* Design time is the time taken to build the classifier from the training data while classification time is the time taken to classify a pattern using the designed classifier. The nearest neighbour classifier does not require any design time. However, the classification time will be high since each test pattern has to be compared to all the patterns in the training set. On the other hand, the neural network classifier requires a high design time to train the weights in the network. But, the classification time will be low as it is only necessary to run the pattern through the trained network to get the class of the pattern.
3. *Space required* If an abstraction of the training set is carried out, the space required will be less. If no abstraction is carried out and the entire training data is required for classification, the space requirement is high. The nearest neighbour classifier requires the entire training data to be stored and therefore, the space required will be more. The neural network, decision tree and the rule-based classifier requires only the abstraction of the training set and therefore requires less space. For instance, the neural network classifier requires only the trained neural network to carry out the classification task. The training data set is not required.
4. *Explanation ability* If the reason for the classifier in choosing the class of a pattern is clear to the user, then its explanation ability is good. For instance, in the decision tree classifier, following the path from the root of the tree to the leaf node for the values of the features in the pattern will give the class of the pattern. Similarly, the user understands why a rule based system chooses a particular class for a pattern. On the other hand, the neural network classifier has a trained neural network and it is not clear to the user what the network is doing.
5. *Noise tolerance* This refers to the ability of the classifier to take care of outliers and patterns wrongly classified.

The accuracy of the classifier is the parameter usually taken into account. However, if the classification time is too large and it is required to carry out the classification task quickly, it may be better to sometimes sacrifice the accuracy and use a classifier which is faster. In places where there is a space limitation like in hand-held devices, it is good to use a classifier requiring little space. In crucial applications like medical diagnostics, explanation ability may be necessary in a classifier so that the doctors are sure that the classifier is doing the right thing.

To estimate how good a classifier is, an estimate can be made using the training set itself. This is known as re-substitution estimate. It assumes that the training data is a good representative of the data. Sometimes, a part of the training data is used as a measure of the performance of the classifier. Usually the training set is divided into smaller sub-sets. One of the sub-sets is used for training while the other is used for validation. The different methods of validation are as follows:

1. *Holdout method* The training set is divided into two sub-sets. Typically two-thirds of the data is used for training and one-thirds is used for validation. It could also be one-half for training and one-half for testing or any other proportion.

2. *Random sub-sampling* In this method, the holdout method is repeated a number of times with different training and validation data each time. If the process is repeated k times, the overall accuracy will be

$$\text{acc}_{\text{overall}} = \frac{1}{k}\Sigma_{i=1}^{k}\text{acc}_i$$

3. *Cross-validation* In cross-validation, each pattern is used the same number of times for training and exactly once for testing. An example of cross-validation is the two-fold cross-validation. Here the data set is divided into two equal sub-sets. First, one set is used for training and the other for testing. Then the roles of the sub-sets are swapped—the set for training becomes the set for validation and vice versa.

   In k-fold cross-validation, the data is divided into k equal sub-sets. During each run, one sub-set is used for testing and the rest of the sub-sets are used for training. This procedure is carried out k times so that each sub-set is used for testing once. A special case of the k-fold cross-validation is when $k = n$, where n is the number of patterns in the data set. This is called the *leave-one-out* approach, where each test set contains only one pattern. The procedure is repeated n times.

4. *Bootstrap procedure* Here a pattern is chosen randomly from the data set, while not deleting it from the data set. Another pattern is then randomly chosen. This is repeated till we have N patterns. These are used for training and the patterns not chosen are used for testing.

2.9 Evaluation of Clustering

It is also necessary to evaluate the clustering carried out by any method. Usually, if the dimensions are low in number, a manual inspection of the clustering graph gives a good idea if the clustering is okay. The quality of the clustering can be measured by examining the similarity between patterns. Patterns belonging to the same cluster should have high similarity and patterns belonging to different clusters should have low similarity.

Discussion

The representation of patterns is very important as a good representation makes use of the discriminating attributes of the objects. Use of good representation reduces the

computation burden in pattern classification. There are a variety of data structures which can be used for representing patterns. The method chosen would depend on the problem. Feature selection is the process of selecting a sub-set of the features which is found to be more relevant and weeding out the features which really do not contribute much to the classification process. All feature selection methods depend on trying out different sub-sets of features and finding the best sub-set. Before deciding on the classifier to be used for a particular task, it is necessary to evaluate the classifier. The criteria to be used for evaluation of classifiers and how to go about evaluating classifiers has been discussed.

Further Reading

Huttenlocher (1993) describes the Hausdorff distance which is used as a similarity measure for patterns with images. The frequent pattern (FP) tree used to represent patterns in a compact way is given by Han et al. (2004).

A number of papers explain the different methods of feature selection. Narendra and Fukunaga (1977) explain the branch and bound algorithm for feature selection. An improvement on this algorithm is given Yu and Yuan (1993). Floating search methods for feature selection are described by Pudil et al. (1994) and Somol et al. (1999). Kuncheva and Jain (1999) and Siedlecki and Sklansky (1989) show how genetic algorithms can be used for feature selection. Kuncheva and Jain (1999) show how we can combine the process of feature selection and prototype selection.

Exercises

1. Find the centroid and medoid (most centrally located pattern) for the following set of patterns:

   (1, 1), (1, 3), (1, 4), (2, 2), (2, 3), (3, 1), (3, 4), (4, 2).

   Under what conditions will the medoid and the centroid be identical?

2. Under what conditions will a set of points give positive minimum variance value? Why is the centroid a good representative of a set of points?

3. Find the edit distance between the two strings ‘‘HOUSE’’ and ‘‘MOUND’’.

4. Show that the squared Euclidean distance is not a metric.

5. For non-binary data, show that $L_1 > L_2 > L_\infty$.

6. The similarity function $D(X, Y)$ between X and Y as given below is a non-metric.

$S(X,Y) = \frac{X^t Y}{||X||\,||Y||}$

$D(X,Y) = 1 - S(X,Y)$

Show how this can be converted into a metric.

7. The Hausdorff distance between two points I and J is given by

   $\max(\max_{i \in I} \min_{j \in J} \parallel i - j \parallel, \max_{j \in J} \min_{i \in I} \parallel i - j \parallel)$

   This distance is not a metric. Which property is violated here, making it a non-metric?

8. Find the mutual neighbourhood distance (MND) between D and E given the following set of points:

   A = (1, 1); B = (2, 2); C = (2, 1); D = (3, 1);

   E = (4, 4); F = (5, 4); G = (6, 4); H = (6, 5)

9. Give the conditions under which the FP tree built on N d-dimensional binary patterns has the minimum number of nodes where each pattern has at most l ($< d$) 1s.

10. A data set consists of the following patterns:

    (1, 1, 1), (2, 2, 1), (1.5, 0.5, 1), (1, 3, 1), (4, 4, 2), (5, 5, 2), (4, 5, 2), (4, 6, 2)

    where each pattern consists of the x-coordinate, y-coordinate and the class. Find the direction of the W vector associated with Fisher's linear discriminant.

11. Given n d-dimensional patterns from two classes and $n < d$, comment on whether S_B and S_W used in Fisher's linear discriminant are singular or not.

12. Find the direction of W using Fisher's linear discriminant when $m_1 = m_2$ and $s_1 = s_2 = 0.5I$, where I is the identity matrix.

13. Given below is a set of patterns with three features X, Y and Z.

X	Y	Z	Output
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Are any of the features redundant? Explain.

14. If there are ten features and it is necessary to reduce the number of features to six so that the best set of six features is chosen, what is the number of feature sub-sets to be evaluated to find the optimal set of six features in the exhaustive search?

Computer Exercises

1. Write a program to generate the FP tree of a set of patterns. Use the program to generate the FP tree for the data given in Table 2.1.
2. Extend the program of Computer Exercise 1 so as to get the nearest neighbour of any transaction using the FP tree and use it on the data given in Table 2.1.
3. Write a program to obtain the Fisher's linear discriminant for a training data set. Obtain the Fisher's linear discriminant for the two-dimensional data set :

   (1, 1, 1), (1, 2, 1), (1, 3, 1), (2, 1, 1), (2, 2, 1), (2, 3, 1), (2, 3.5, 1), (2.5, 2,1), (3.5, 1, 1), (3.5, 2, 1), (3.5, 3, 2), (3.5, 4,2), (4.5, 1, 2), (4.5, 2, 2), (4.5, 3, 2), (5, 4, 2), (5, 5, 2), (6, 3, 2), (6, 4, 2), (6, 5, 2)

   where each pattern is represented by feature 1, feature 2 and the class.
4. Write a program to implement the principal component analysis and use it on the data given in Computer Exercise 3.
5. Implement the branch and bound search for feature selection. Apply it to a data set having a number of features and find the set of features found by the program.
6. Implement the sequential floating forward search and apply it to a data set used for Computer Exercise 5. Compare the features found with that found using the branch and bound algorithm.
7. Implement the sequential floating backward search and apply it to the data set used for Computer Exercise 5. Compare the feature set found with that found using SFFS and branch and bound algorithm. Which algorithm gives the best set of features?

Bibliography

1. Han, Jaiwei, Jian Pei, Yiwen Yin and Runying Mao. Mining frequent patterns without candidate generation: A frequent pattern tree approach. *Data Mining and Knowledge Discovery* 8(1): 53–87. 2004.

2. Huttenlocher, D. P., G. A. Klanderman and W. A. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 15(9): 850–863. 1993.

3. Jain, A. K. and D. Zongker. Feature selection: Evaluation, application and small sample performance. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 19:153–157. 1997.

4. Kuncheva, L. and L. C. Jain. Nearest neighbor classifier: Simultaneous editing and feature selection. *Pattern Recognition Letters* 20:1149–1156. 1999.

5. Narendra, P. M. and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Trans. Computers* 26(9): 917–922. 1977.

6. Pudil, P., J. Novovicova and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters* 15: 1119–1125. 1994.

7. Siedlecki, W. and J. Sklansky. A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters* 10: 335–347. 1989.

8. Somol, P., P. Pudil, J. Novovicova, P. Paclik. Adaptive floating search methods in feature selection. *Pattern Recognition Letters* 20: 1157–1163. 1999.

9. Yu, Bin and Baozong Yuan. A more efficient branch and bound algorithm for feature selection. *Pattern Recognition* 26(6): 883–889. 1993.

3 Nearest Neighbour Based Classifiers

Learning Objectives

This chapter is an introduction to nearest neighbour classifiers. At the end of the chapter, you will understand:

- What a nearest neighbour(NN) algorithm is
- The different variants of NN algorithms like
  - The k nearest neighbour(kNN) algorithm
  - The modified k nearest neighbour (MkNN) algorithm
  - The fuzzy kNN algorithm
  - The r near algorithm
- The use of efficient algorithms
- What is meant by data reduction
- The different methods of prototype selection used in classification like
  - The minimal distance classifier(MDC)
  - The condensed nearest neighbour(CNN) algorithm
  - The modified condensed nearest neighbour (MCNN) algorithm
  - Editing methods

One of the simplest decision procedures that can be used for classification is the nearest neighbour (NN) rule. It classifies a sample based on the category of its nearest neighbour. When large samples are involved, it can be shown that this rule has a probability of error which is less than twice the optimum error—hence there is less than twice the probability of error compared to any other decision rule. The nearest neighbour based classifiers use some or all the patterns available in the training set to classify a test pattern. These classifiers essentially involve finding the similarity between the test pattern and every pattern in the training set.

3.1 Nearest Neighbour Algorithm

The nearest neighbour algorithm assigns to a test pattern the class label of its closest neighbour. Let there be n training patterns, $(X_1, \theta_1), (X_2, \theta_2), ..., (X_n, \theta_n)$, where X_i is

of dimension d and θ_i is the class label of the ith pattern. If P is the test pattern, then if

$$d(\mathrm{P}, \mathrm{X}_k) = \min\{d(\mathrm{P}, \mathrm{X}_i)\}$$

where $i = 1 \ldots n$. Pattern P is assigned to the class θ_k associated with X_k.

EXAMPLE 1

Let the training set consist of the following three dimensional patterns:

$$\begin{array}{lll}
\mathrm{X}_1 = (0.8, 0.8, 1), & \mathrm{X}_2 = (1.0, 1.0, 1), & \mathrm{X}_3 = (1.2, 0.8, 1) \\
\mathrm{X}_4 = (0.8, 1.2, 1), & \mathrm{X}_5 = (1.2, 1.2, 1), & \mathrm{X}_6 = (4.0, 3.0, 2) \\
\mathrm{X}_7 = (3.8, 2.8, 2), & \mathrm{X}_8 = (4.2, 2.8, 2), & \mathrm{X}_9 = (3.8, 3.2, 2) \\
\mathrm{X}_{10} = (4.2, 3.2, 2), & \mathrm{X}_{11} = (4.4, 2.8, 2), & \mathrm{X}_{12} = (4.4, 3.2, 2) \\
\mathrm{X}_{13} = (3.2, 0.4, 3), & \mathrm{X}_{14} = (3.2, 0.7, 3), & \mathrm{X}_{15} = (3.8, 0.5, 3) \\
\mathrm{X}_{16} = (3.5, 1.0, 3), & \mathrm{X}_{17} = (4.0, 1.0, 3), & \mathrm{X}_{18} = (4.0, 0.7, 3)
\end{array}$$

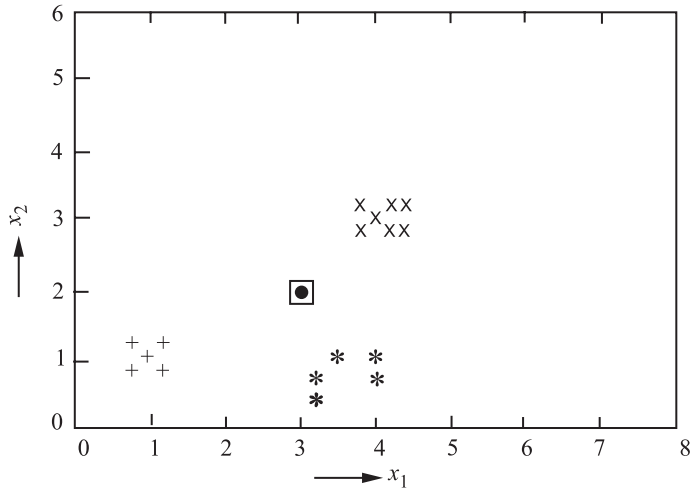


Figure 3.1 Example data set

For each pattern, the first two numbers in the triplets gives the first and second features, and the third number gives the class label of the pattern.

This can be seen plotted in Figure 3.1. Here ''+'' corresponds to Class 1, ''X'' corresponds to Class 2 and ''*'' corresponds to Class 3.

Now if there is a test pattern P = (3.0, 2.0), it is necessary to find the distance from P to all the training patterns.

Let the distance between X and P be the Euclidean distance

$$d(\mathrm{X}, \mathrm{P}) = \sqrt{(X[1] - P[1])^2 + (X[2] - P[2])^2}$$

The distance from a point P to every point in the set can be computed using the above formula. For P = (3.0, 2.0), the distance to X_1 is

$$d(\mathrm{X}_1, \mathrm{P}) = \sqrt{(0.8 - 3.0)^2 + (0.8 - 2.0)^2} = 2.51$$

We find, after calculating the distance from all the training points to P, that the closest neighbour of P is X_{16}, which has a distance of 1.12 from P and belongs to Class 3. Hence P is classified as belonging to Class 3.

3.2 Variants of the NN Algorithm

3.2.1 k-Nearest Neighbour (kNN) Algorithm

In this algorithm, instead of finding just one nearest neighbour as in the NN algorithm, k neighbours are found. The majority class of these k nearest neighbours is the class label assigned to the new pattern. The value chosen for k is crucial. With the right value of k, the classification accuracy will be better than that got by using the nearest neighbour algorithm.

EXAMPLE 2

In the example shown in Figure 3.1, if k is taken to be 5, the five nearest neighbours of P are X_{16}, X_7, X_{14}, X_6 and X_{17}. The majority class of these five patterns is class 3.

This method will reduce the error in classification when training patterns are noisy. The closest pattern of the test pattern may belong to another class, but when a number of neighbours are obtained and the majority class label is considered, the pattern is more likely to be classified correctly. Figure 3.2 illustrates this.

EXAMPLE 3

It can be seen from Figure 3.2 that the test point P is closest to point 5 which is an outlier in Class 1 (represented as a cross). If kNN algorithm is used, the point P will be classified as belonging to Class 2 represented by circles.

Choosing k is crucial to the working of this algorithm. For large data sets, k can be larger to reduce the error. The value of k can be determined by experimentation, where a number of patterns taken out from the training set(validation set) can be classified using the remaining training patterns for different values of k. It can be chosen as the value which gives the least error in classification.

EXAMPLE 4

In Figure 3.1, if P is the pattern (4.2, 1.8), its nearest neighbour is X_{17} and it would be classified as belonging to Class 3 if the nearest neighbour algorithm is used. If

the 5 nearest neighbours are taken, it can be seen that they are X_{17} and X_{16}, both belonging to Class 3 and X_8, X_7 and X_{11}, belonging to Class 2. Following the majority class rule, the pattern would be classified as belonging to Class 2.

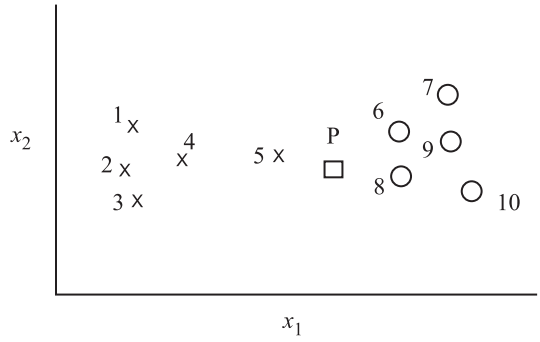


Figure 3.2 P can be correctly classified using the kNN algorithm

3.2.2 Modified k-Nearest Neighbour (MkNN) Algorithm

This algorithm is similar to the kNN algorithm, inasmuch as it takes the k nearest neighbours into consideration. The only difference is that these k nearest neighbours are weighted according to their distance from the test point. It is also called the distance-weighted k-nearest neighbour algorithm. Each of the neighbours is associated with the weight w which is defined as

$$w_j = \begin{cases} \frac{d_k - d_j}{d_k - d_1} & \text{if } d_k \neq d_1 \\ 1 & \text{if } d_k = d_1 \end{cases}$$

where $j = 1, .., k$. The value of w_j varies from a maximum of 1 for the nearest neighbour down to a minimum of zero for the most distant. Having computed the weights w_j, the MkNN algorithm assigns the test pattern P to that class for which the weights of the representatives among the k nearest neighbours sums to the greatest value.

Instead of using the simple majority rule, it can be observed that MkNN employs a weighted majority rule. This would mean that outlier patterns have lesser effect on classification.

EXAMPLE 5

Consider P = $(3.0, 2.0)$ in Figure 3.1. For the five nearest points, the distances from P are

$d(\mathrm{P}, \mathrm{X}_{16}) = 1.2$; $d(\mathrm{P}, \mathrm{X}_7) = 1.13$; $d(\mathrm{P}, \mathrm{X}_{14}) = 1.32$;

$d(\mathrm{P}, \mathrm{X}_6) = 1.41$; $d(\mathrm{P}, \mathrm{X}_{17}) = 1.41$;

The values of w will be

$$w_{16} = 1$$

$$w_7 = \frac{(1.41 - 1.13)}{(1.41 - 1.12)} = 0.97$$

$$w_{14} = \frac{(1.41 - 1.32)}{(1.41 - 1.12)} = 0.31$$

$$w_6 = 0$$

$$w_{17} = 0$$

Summing up for each class, Class 1 sums to 0, Class 2 to which X_7 and X_6 belong sums to 0.97 and Class 3 to which X_{16}, X_{14} and X_{17} belong sums to 1.31. Therefore, the point P belongs to Class 3.

It is possible that kNN and MkNN algorithms assign the same pattern a different class label. This is illustrated in the next example.

EXAMPLE 6

In Figure 3.1, when P = (4.2, 1.8), the five nearest patterns are X_{17}, X_8, X_{11}, X_{16} and X_7. The distances from P to these patterns are

$d(\mathrm{P}, \mathrm{X}_{17}) = 0.83$; $d(\mathrm{P}, \mathrm{X}_8) = 1.0$; $d(\mathrm{P}, \mathrm{X}_{11}) = 1.02$

$d(\mathrm{P}, \mathrm{X}_{16}) = 1.06$; $d(\mathrm{P}, \mathrm{X}_7) = 1.08$

The value of w will be

$$w_{17} = 1$$

$$w_8 = \frac{(1.08 - 1.0)}{(1.08 - 0.83)} = 0.32$$

$$w_{11} = \frac{(1.08 - 1.02)}{(1.08 - 0.83)} = 0.24$$

$$w_{16} = \frac{(1.08 - 1.06)}{(1.08 - 0.83)} = 0.08$$

$$w_7 = 0$$

Summing up for each class, Class 1 sums to 0, Class 2 to which X_8, X_{11} and X_7 belong sums to 0.56 and Class 3 to which X_{17} and X_{16} belong sums to 1.08 and therefore, P is classified as belonging to Class 3. Note that the same pattern is classified as belonging to Class 2 when we used the k nearest neighbour algorithm with $k = 5$.

3.2.3 Fuzzy kNN Algorithm

In this algorithm, the concept of fuzzy sets, whose elements have a degree of membership, is used. In classical set theory, an element either belongs or does not belong to a set. In fuzzy sets, the elements of the set have a membership function attached to them which is in the real unit interval [0, 1].

In fuzzy kNN algorithm, the fuzzy sets are the c classes. Each pattern belongs to every class i with a membership value which depends on the class of its k neighbours.

The fuzzy kNN algorithm finds the k nearest neighbours of the test point and assigns a class membership for every point to all the classes. A new sample P has membership $\mu_i(\mathrm{P})$ for class i which is given by

$$\mu_i(\mathrm{P}) = \frac{\Sigma_{j=1}^{K} \mu_{ij} \left(\frac{1}{d(\mathrm{P},\mathrm{X}_j)^{\frac{2}{m-1}}} \right)}{\Sigma_{j=1}^{K} \left(\frac{1}{d(\mathrm{P},\mathrm{X}_j)^{\frac{2}{m-1}}} \right)}$$

μ_{ij} gives the membership in the ith class of the jth vector of the training set. One way of assigning this membership is to give complete membership in their known class and zero membership in all other classes. Another way to assign this membership is to base it on the distance of the sample from the class mean for all the classes. Here m is a constant and its value is provided by the user.

Each pattern has a membership of 1 to the classes which figure in the k nearest neighbours and a membership of 0 to the other classes. The test pattern is given a membership to every class and the class to which it has the highest membership value is the one it is assigned to.

EXAMPLE 7

In the example shown in Figure 3.1, if we take P = (3.0, 2.0) and k = 5, the 5 nearest neighbours will be 16, 7, 14, 6 and 17. When we consider Class 1, since there are no patterns from Class 1, $\mu_1(P)$ will be 0. For Class 2, μ_{ij} will be 1 for patterns 6 and 7, and 0 for patterns 16, 14 and 17. For Class 3, μ_{ij} will be 1 for patterns 16, 14 and 17, and 0 for patterns 6 and 7. Taking $m = 2$, we get

$\mu_1(\mathrm{P})$ = 0

$$\mu_2(\mathrm{P}) = \frac{\frac{1}{1.414^2} + \frac{1}{1.131^2}}{\frac{1}{1.414^2} + \frac{1}{1.131^2} + \frac{1}{1.118^2} + \frac{1}{1.315^2} + \frac{1}{1.414^2}} = 0.406$$

$$\mu_3(\mathrm{P}) = \frac{\frac{1}{1.118^2} + \frac{1}{1.315^2} + \frac{1}{1.414^2}}{\frac{1}{1.118^2} + \frac{1}{1.315^2} + \frac{1}{1.414^2} + \frac{1}{1.414^2} + \frac{1}{1.131}} = 0.594$$

Therefore, we assign the pattern P to class 3.

3.2.4 *r* Near Neighbours

Instead of looking at nearest neighbours, we can also look at all the neighbours within some distance r of the point of interest. The algorithm is as follows.

STEP 1: Given the point P, determine the sub-set of data that lies in the ball of radius r centred at P,
$B_r(\mathrm{P}) = \{\mathrm{X}_i \in \mathrm{X} \text{ s.t. } ||\mathrm{P} - \mathrm{X}_i|| \leq r\}$

STEP 2: If $B_r(\mathrm{P})$ is empty, then output the majority class of the entire data set.

STEP 3: If $B_r(\mathrm{P})$ is not empty, output the majority class of the data points in it.

If the nearest neighbour of the point of interest is very far away, then that neighbour would have little to do with the point. It is from this logic that we get the r near neighbours algorithm.

This algorithm can be used to identify outliers. If a pattern does not have any similarity with the patterns within the radius chosen, it can be identified as an outlier. The choice of the radius r is crucial to the algorithm.

EXAMPLE 8

In the case of the example shown in Figure 3.1, from the point P = (3.0, 2.0), patterns which are in a radius of 1.45, are X_6, X_7, X_8, X_9, X_{14}, X_{16} and X_{17}. The majority of these patterns belong to Class 2. P is therefore assigned to Class 2.

3.3 Use of the Nearest Neighbour Algorithm for Transaction Databases

Data collected from scientific experiments, or monitoring of physical systems such as telecommunications networks or from transactions at a supermarket, are stored in transaction databases. It is also called market-basket data and consists of transactions

made by each customer. In the case of supermarket data, each transaction contains items bought by a customer. Each transaction in the database could be of a different size. The goal of using this data is to see if occurrence of certain items in the transactions can be used to deduce the occurrence of other items. In other words, it is necessary to find the associative relationship between items. This is called association rule mining. To simplify and speed up the process, only items which occur frequently are considered. A minimum support value is chosen and items which occur less than the minimum support are removed. These transaction databases can be represented using the FP tree described in Section 2.1.5.

The FP tree can be used to find the nearest neighbour to a test pattern in a transaction data. The following procedure is used :

STEP 1: Remove items in the test pattern which are below the minimum support .

STEP 2: Arrange the remaining items in order according to that in the FP tree.

STEP 3: Search from the root node for the branch having the first item in the test pattern. If it exists, look for the next item in the test pattern and so on. When the item does not exist, then look at all the branches from that point and choose the branch having maximum items in common with the test pattern. This gives the nearest neighbour.

EXAMPLE 9

Consider Example 3 in Chapter 2. Figure 2.7 shows the FP tree for the transaction database shown in Table 2.1. Now consider a test pattern a, b, c, d, f, g, h, l, p.

Removing items which are below the minimum support, we get a, d, h, l, p. Arranging these items according to the items in the FP tree, we get l, a, d, p, h. Starting from the root node of the FP tree l, if we compare the items remaining in the test pattern, we can see that it has maximum items common with 7 and therefore can be classified as belonging to digit 7.

3.4 Efficient Algorithms

The nearest neighbour classification method is very time consuming especially when the number of training patterns is large. To overcome this, many efficient algorithms have been proposed which aim to find the nearest neighbour quickly. Some of these algorithms are detailed below. Many of them entail using a pre-processing step which requires additional computing time, but has to be done only once however large the number of test vectors to be classified is. Pre-processing might require a preliminary

ordering of the prototypes based on intra-set distances, together with an application of the triangle inequality during the examination stage. This might require considerable time but since it needs to be done only once, it does save time in the long run.

3.4.1 The Branch and Bound Algorithm

If the data are stored in an ordered fashion, it is possible that we may not have to look at all the points to find the nearest neighbour. For example, if the data is stored in a tree-like data structure, the search for the nearest neighbour can be performed efficiently by not having to search down branches where one can obtain a lower bound for the distance that is higher than the current best value found. The data is clustered into representative groups S_j, where each group has the smallest possible radius. Associated with each cluster is the centre of the cluster μ_j and the radius of the cluster, r_j. After searching one of the clusters, point X^* is found which is the nearest neighbour from the cluster to the point of interest P. Let this distance be $d(\mathrm{P}, \mathrm{X}_j^*) = d$. Lower bounds can be computed for the distances in the other clusters. Consider the point $\mathrm{X}_k \in \mathrm{S}_j$, then by triangular inequality,

$$d(\mathrm{P}, \mu_j) \leq d(\mathrm{P}, \mathrm{X}_k) + d(\mathrm{X}_k, \mu_j) \leq d(\mathrm{P}, \mathrm{X}_j) + r_j$$

Therefore, a lower bound b_j can be computed for the distance between any $\mathrm{X}_k \in S_j$ and P as follows:

$$\min_{\mathrm{X}_k \in \mathrm{S}_j} d(\mathrm{P}, \mathrm{X}_k) \geq b_j = d(\mathrm{P}, \mu_j) - r_j$$

Clusters which satisfy the inequality $b_j \geq d$ need not be searched. If each cluster has an exactly analogous cluster structure within the clusters themselves, we get the following recursive algorithm.

STEP 1: Cluster the data points into L sets S_i, $i = 1$ to L. These are first level clusters. For each cluster, maintain the centre μ_i and the radius r_i. Within each cluster, S_i, cluster the data into L sub-clusters $S_{i,j}$ for $j = 1$ to L. Maintain the centre $\mu_{i,j}$ and the radius $r_{i,j}$. These are second level clusters. Continue this recursive clustering until there are clusters of one point.

STEP 2: To find the nearest neighbour to a new point P,

1. *Branch step* First compute b_j. Then find the nearest neighbour in the cluster with the smallest b_j. If the clusters have clusters, this step is done recursively.
2. *Bound step* If that cluster does not satisfy the bound, look into the cluster with the next highest b_j. Otherwise stop and output the result.

It can be shown that on an average, there is considerable improvement over the standard nearest neighbour algorithm.

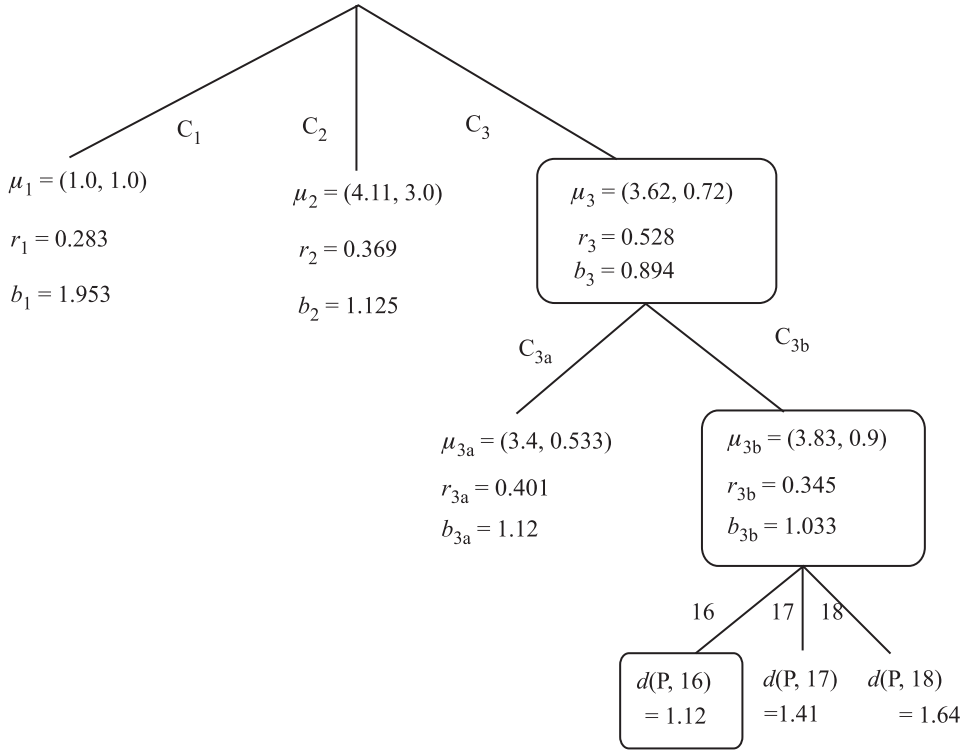


Figure 3.3 Working of the branch and bound algorithm for clustering

EXAMPLE 10

Let us take the example in Figure 3.1. The working of the algorithm is shown in Figure 3.3. Clustering of the points has to be first carried out. Let the points of Class 1 be taken as one cluster, points of Class 2 as the second cluster and the points of Class 3 as the third cluster. Further Cluster 1 is sub-clustered as Cluster 1a and 1b, Cluster 2 is sub-clustered as Cluster 2a and 2b and Cluster 3 is sub-clustered as 3a and 3b. At the next level, each point is taken to be a sub-cluster. This clustering of the points is shown in Figure 3.4.

The centre of Cluster 1 is (1.0, 1.0) and the radius is 0.283. The centre of Cluster 2 is (4.11, 3.0) and the radius is 0.369. The centre of Cluster 3 is (3.62, 0.72) and the radius is 0.528. Taking a point P = (3.0, 2.0), at the first level, b_j for each cluster j can be found. $b_1 = d(\text{P}, \mu_1) - r_1 = 1.953$. Similarly $b_2 = 1.125$ and $b_3 = 0.894$. Since b_3 is the smallest, the sub-clusters of Cluster 3 are searched, and we get the

centre of Cluster 3a as (3.4, 0.533) and the radius to be 0.401. The centre of Cluster 3b is (3.83, 0.9) and the radius 0.345. This gives rise to $b_{3a} = 1.12$ and $b_{3b} = 1.033$. The second sub-cluster of Cluster 3 is therefore searched and point 16 is found to be the point closest to point P. The bound d is calculated as the distance from P to point 16 which is 1.12 . Since b_1 and b_2 are greater than d, the clusters 1 and 2 need not be searched and therefore X_{16} is the closest point to P. X_{16} is declared as the nearest neighbour of P.

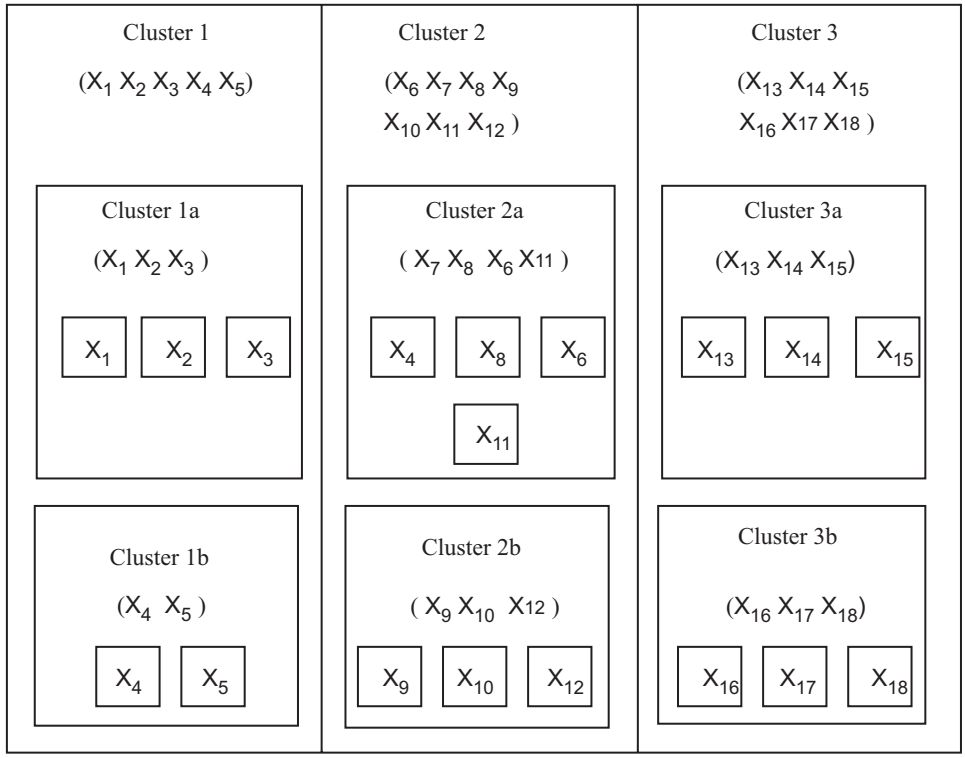


Figure 3.4 Clustering of points using branch and bound method

3.4.2 The Cube Algorithm

In this algorithm, during the pre-processing stage, the patterns are projected onto each axis and a side length l for the initial hypercube to be inspected is specified. Figure 3.5 illustrates in two dimensions the square C of side l centered at P = (p_1, p_2) and its projection onto both axes. By appropriately incrementing or decrementing l in an iterative manner, a hypercube containing exactly k points can be quickly found. The algorithm is described below:

STEP 1: Form $B_i(j)$ and $K_i(j)$, i = 1,..., d and $j = 1,..., n$, where d is the number of dimensions (or features) and n gives the number of patterns. B_i is a real array containing the n ordered values of feature i. K_i is an integer array of indexes, where $K_i(J)$ contains the index of the pattern represented by $B_i(j)$. Let the integer array Z contain n elements each of which is initialised to 1.

STEP 2: Let P = $(p_1, ..., p_d)$ be the test vector.

STEP 3: For each axis i, set pointer ptr_{i1} to the location in A_i of the first element $\geq p_i - \frac{l}{2}$; set pointer ptr_{i2} to the location in B_i of the last element $\leq p_i + \frac{l}{2}$.

STEP 4: For each axis i, go sequentially through the array K_i starting with $K_i(P1_i)$ and ending with $K_i(\text{ptr}_{i2})$. At each location in K_i, shift left 1 bit, the contents of the cell in Z pointed to by that location. In other words, at each location in K_i, multiply by 2, the contents of the cell pointed to by that location.

STEP 5: Count the locations in Z with the value 2^d, and let L equal this count. This count gives the points which fall within the hypercube.

STEP 6: If $L = k$, stop. If $L < k$ (where k = number of neighbours required), increase l appropriately. Reset pointers, repeat the scanning and shifting procedure only on the newly included cells of K_i, and go to Step 4. If $L > k$, decrease l appropriately. Reset pointers, scan only the newly excluded cells of K_i and shift right 1 bit, the proper cells of Z. Go to Step 4.

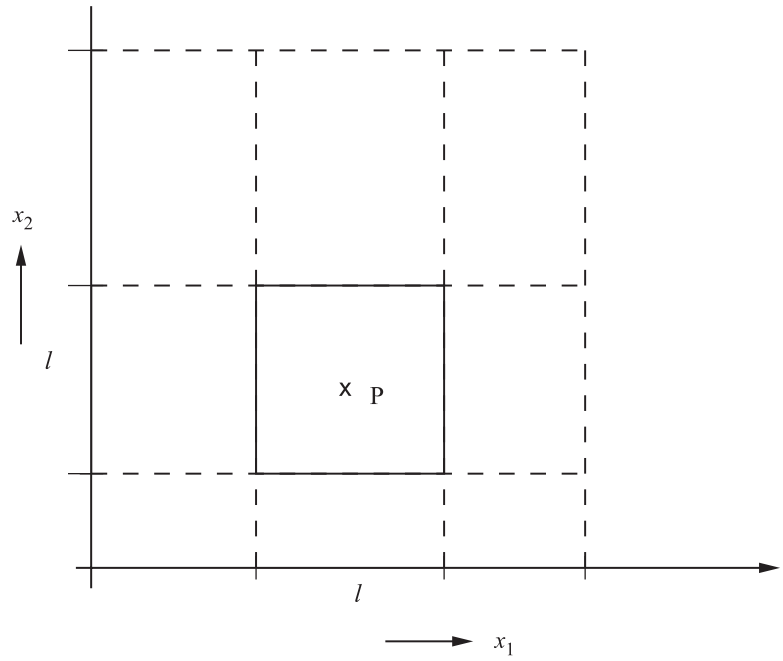


Figure 3.5 Projection of a hypercube

EXAMPLE 11

Figure 3.6 shows a set of two-dimensional points. The points 1, 2, 3, 4, 5 and 6 belong to Class 1 and the points 7, 8, 9, 10, 11 and 12 belong to Class 2. It is required to find k nearest neighbours of P, where $k = 5$. If l is chosen to be 2, the points that fall within the square will obey the conditions $2 \leq p_1 \leq 4$ and $2 \leq p_2 \leq 4$. It is found that 3 points fall within the square. l is then increased by 0.5 and it is found that 5 points fall within the square which give the 5 nearest neighbours. The majority class labels of these five points can be used to find the class label of the point P. Since 4, 5 and 6 belong to Class 1 and 7 and 9 belong to Class 2, among the 5 closest patterns, three belong to Class 1 and two belong to Class 2. P is therefore assigned to Class 1.

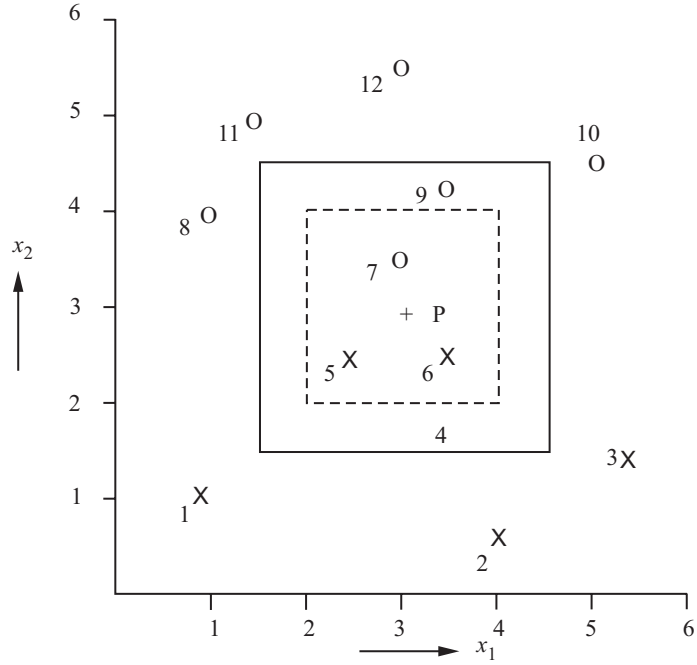


Figure 3.6 Example of clustering using the cube algorithm

This algorithm requires a much steeper cost for storage. It requires the storage of N sorted feature values and their indexes for each axis, or $2dN$ words. Processing time involves the sorting of dN word arrays which is of the order of $dN \log N$ comparisons.

3.4.3 Searching for the Nearest Neighbour by Projection

Various projection algorithms are available for performing nearest neighbour searches. In the simplest case of one-dimensional space, if the point set is ordered, we can find

the nearest neighbours of point P by locating P's position in the order. Finding its successor and predecessor and taking the closest of these two points, will give the nearest neighbour.

In the two-dimensional case, the points are sorted in increasing order using the first coordinate value. To find the nearest neighbour to point P, we locate P's position in the sorted list of the first coordinate value. We then search out the nearest neighbours of P. We examine the point that is closer to P in x-distance first and remember the closest point to P so far observed (which we will call Q). The search can be pruned as soon as we know that any other point we would visit is necessarily further from P than Q is. Specifically, if only the distance in the x-direction from P to the next point to be searched is greater than the distance to Q, then the search can be stopped and Q can be reported as P's nearest neighbour. Figure 3.7 shows this graphically.

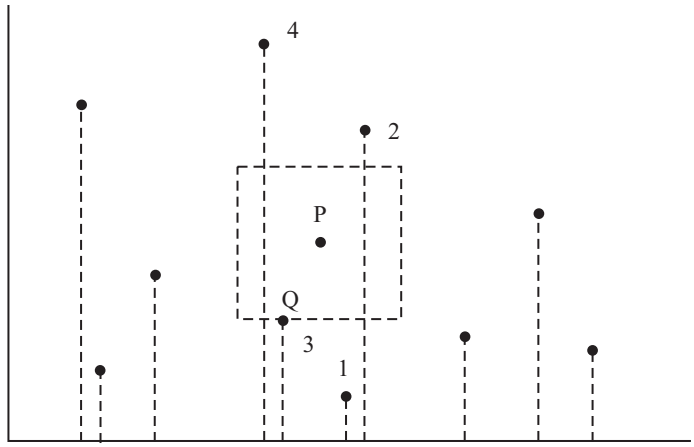


Figure 3.7 Searching for the nearest neighbour by projection on the x-axis

This process is modified to project the points onto both the x- and y-axes, instead of only the x-axis. Now the search is carried out both in the x- and y-axes in parallel. In other words, the first step is on the x-axis, the second on the y-axis, the third on the x-axis etc. The search stops as soon as the search in either direction stops. Figure 3.8 shows the nearest neighbour search using two projections.

EXAMPLE 12

In the example given in Figure 3.1, if the test point is P = (3, 2), all the data points are arranged according to the value in the x-axis. The order in increasing order of x-axis values is X_{13}, X_{14}, X_{16}, X_7, X_9, X_6, X_{17}, X_{18}, X_8, X_{10} etc. The distance of these points in the x-direction is 0.2, 0.2, 0.5, 0.8, 0.8, 1.0, 1.0, 1.0, 1.2 and 1.2. These points are taken in the order given, and their actual distance to P is found. At each

point, the closest point to P so far is kept in memory. The distances from X_{13}, X_{14}, X_{16}, X_7, X_9, X_{15}, X_6, X_{17} and X_{18} to P are 1.162, 1.315, 1.118, 1.131, 1.44, 1.7, 1.414, 1.414 and 1.64 . Starting from X_{13}, the actual distance is checked and the minimum so far is noted. When X_{16} is reached, the minimum distance changes to 1.118. When X_{18} is reached, the closest point is X_{16} with a distance of 1.118 . The next point X_8 has a distance in the x-direction of 1.2 which is larger than the actual distance of the closest point X_{16}. Then it is not necessary to check any more points, and X_{16} is the closest point to P.

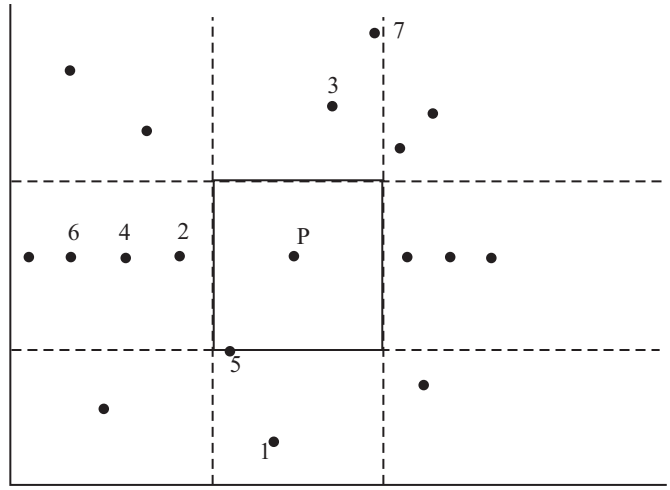


Figure 3.8 Searching for the nearest neighbour using two projections

3.4.4 Ordered Partitions

A pre-processing technique called the ordered partition method is used to reduce the computational requirements for finding the k nearest neighbours to a test sample among n training samples in a d-dimensional feature space. The training patterns are partitioned by their first coordinate values along each axis such that the ordering characteristic is preserved between the partitions. The width of the partitions (or blocks) are adjusted so that each partition contains the same number of samples.

The search starts from level 0 and the squared distance r_l^2 is found for each branch of the tree. This distance is defined recursively as

$$r_l^2 = \begin{cases} r_{l-1}^2 + \min((x_l - a_l)^2, (x_l - b_l)^2), & \text{for } l \neq d \\ r_{d-1}^2 + (x_d - a_d)^2, & \text{for } l = d \end{cases}$$

At each level l, the squared distance r_l^2 is found from the test sample. The node with the smallest distance at that level is chosen to continue the search. When a terminal

node is found, that distance is noted and the smallest distance d is kept in memory. Any node whose distance is greater than d at any level need not be searched.

This procedure finds the k nearest neighbours in a constant expected time.

EXAMPLE 13

Consider the following three-dimensional patterns:

$X_1 = (1, 1, 1)$ $X_2 = (1.2, 2.0, 4.0)$ $X_3 = (2.0, 1.5, 3)$
$X_4 = (2.5, 3.0, 5.0)$ $X_5 = (3.0, 7.0, 6.0)$ $X_6 = (3.5, 2.5, 3.5)$
$X_7 = (4.0, 6.0, 2.5)$ $X_8 = (4.5, 5.5, 4.5)$ $X_9 = (5.0, 1.5, 2.0)$
$X_{10} = (5.5, 6.5, 1.5)$ $X_{11} = (6.0, 8.0, 7.5)$ $X_{12} = (7.0, 9.0, 8.0)$

Figure 3.9 shows the partitioned data when these patterns are first partitioned according to the first dimension, then the second dimension and then the third dimension.

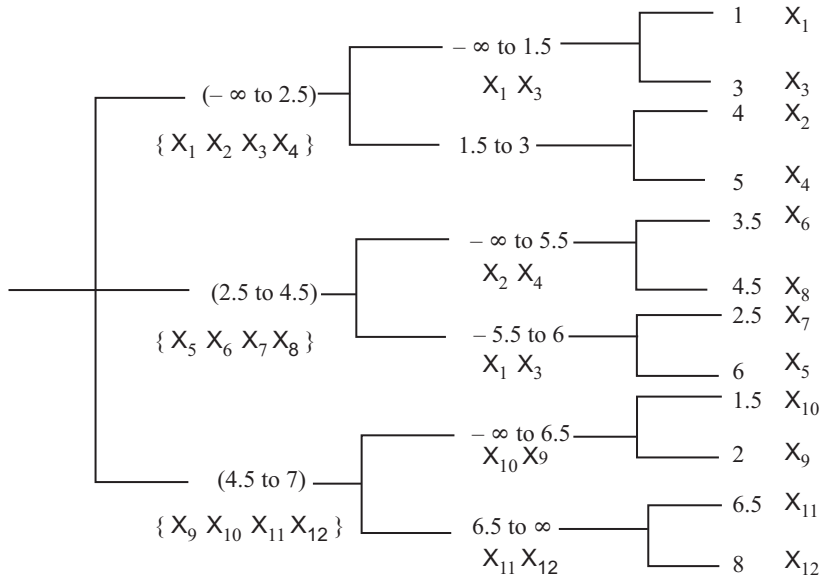


Figure 3.9 Partitioning of data for search using ordered partitions

If the point P is (7.0, 7.0, 7.0), searching through the partitioned data, first the last branch, i.e., the first dimension being between 4.5 to 7 is chosen. From there, the second branch is chosen for the second dimension, i.e., 6.5 to ∞. From there, X_{11} is chosen as the closest point up to that point. This will give a distance of 0 + 0.25 + 0.25 = 0.5 . The other branch in the second dimension namely $-\infty$ to 6.5 is also searched. But the other two branches in the first dimension are not searched

as the squared distance in the first level itself exceeds 0.5 . This distance is 20.25 for the first branch and 6.25 for the second branch. Therefore, X_{11} is the closest point to point P.

3.4.5 Incremental Nearest Neighbour Search

This incremental search algorithm finds the next nearest neighbour more efficiently by eliminating redundant computations. The collection of n training data is stored in a k-d tree. The k-d tree is a binary tree structure for storing and performing operations on data containing k-dimensional keys. The root of the k-d tree represents the k-dimensional space containing the collection of data. Each node represents a sub-space containing a sub-set of the collection of data. Partitioning a non-terminal node's sub-space into two parts by a hyper-plane perpendicular to one of the k coordinate axes gives the two children of the node. The position of the hyper-plane is chosen so that each of the children contains approximately half the parents' data patterns. The discriminating axis is determined by choosing the axis along which the parent's data are most dispersed. Children that contain less than a threshold number of data points become leaf nodes of the tree. Each leaf node is represented in the k-d tree as a hyper-volume. Figure 3.10 shows the construction of the k-d tree for two-dimensional data. Here each leaf node is represented by a rectangle in the k-d tree.

The nearest neighbour search using the k-d tree is carried out by first locating the test pattern in the k-d tree created using the training data. By descending the tree, the leaf node which spatially contains the test point is located. The distance of all the data points in the leaf node from the test point is found and ordered in increasing order. The data point closest to the test point is a possible candidate for the nearest neighbour. The distance r from the test point to this data point is an upper bound for the distance to the actual nearest neighbour. If we draw a hyper-sphere S_r centered at the test point with radius r and if S_r is completely enclosed by the hyper-volume of the leaf node, then only the points in the leaf node need be searched to find the nearest neighbour. If S_r is not entirely within the leaf node, the tree is ascended one level and then descended one level to the other children. Only leaf nodes which spatially intersect S_r need to be considered. Here again the distance to all the data points in this leaf node are computed and the data point which is closest in these distances and those computed for the earlier leaf node are considered. The closest point defines the new value of r. This process of ascending the tree and evaluating new data points is continued till S_r is entirely enclosed by the space defined by the current non-terminal node.

EXAMPLE 14

Figure 3.10 shows a k-d tree drawn for the set of points. The closest neighbour for the point P is found within the same leaf node and a circle drawn. Since this circle

lies completely within the square which represents a leaf node, the point found is the closest neighbour. In the case of the point Q, the circle also includes other leaf nodes. It is therefore necessary to search those leaf nodes also to find the nearest neighbour.

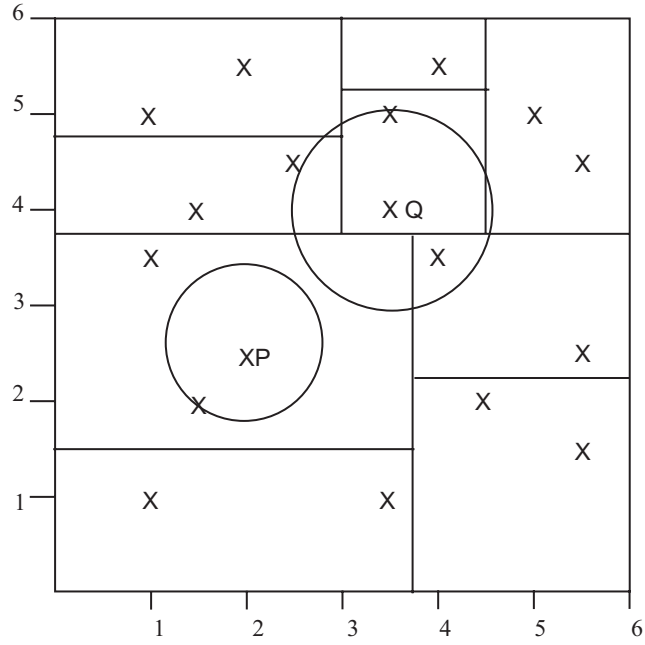


Figure 3.10 Construction of the k-d tree and search for the closest node

3.5 Data Reduction

In supervised learning, the efficacy of the classification is dependent on the training data. A larger training data may lead to better classification but may require a longer time for the classification process. While using classification tools such as neural networks or decision trees, it is only the design time which increases. Once the design is complete, classification of new samples will be fast. But in the case of techniques based on the nearest neighbour rule where every new sample has to be compared to all the training patterns, the time to classify each sample is large. In such cases, reducing the number of training patterns will definitely lead to faster classification. This reduction has to be carried out without sacrificing the classification accuracy.

Another option to reduce classification time is feature selection where the dimensionality of the patterns is reduced so that a sub-set of the features which gives good classification accuracy is used.

Let $F = \{f_1, ..., f_d\}$ be the set of features describing objects as d-dimensional vectors in R^n and let $X = \{X_1, ..., X_n\}$, $X_j \epsilon R^n$ be the data set. Associated with each X_j, $j = 1, ..., n$, is a class label from the set $C = 1, ..., c$. Prototype selection and feature selection work in the following way. For prototype selection, instead of X, we use a sub-set $S_1 \subset \mathrm{X}$ and for feature selection, instead of F, we use $S_2 \subset F$. Feature selection has already been described in Chapter 2. Prototype selection is described in the next section.

3.6 Prototype Selection

Given a set of samples, a prototype would be a sample which would represent or be a typical sample for a large number of samples in the set. For example, given a set of samples, the centroid of the set could be called a prototype of the set. Instead of a single sample, we can have a number of prototypes which collectively represent a set of samples. Prototype selection essentially entails finding the sample or set of samples to represent a larger set of samples.

In spite of the quality of performance of the nearest neighbour rule, it has some practical disadvantages. The performance of the nearest neighbour algorithm increases with increase in the number of training patterns. However, if the training set is too large, use of the nearest neighbour and related techniques will require each test sample to be compared with every sample in the training set. Time and space requirements increase linearly with increase in the number of training patterns. The computational burden is significant and makes this method inapplicable for large training sets. Prototype selection provides a solution to this problem.

Prototype selection refers to the process of reducing the training set used for classification. This can either mean that a representative sub-set of the training patterns is selected or some prototypes based on the training set are chosen.

Formally, let $\chi = \{(\mathrm{X}_1, \theta_1), (\mathrm{X}_2, \theta_2), ..., (\mathrm{X}_n, \theta_n)\}$ be the given labelled training set. Prototype selection refers to the process of obtaining $\chi' = \{\mathrm{X}^1, \mathrm{X}^2, ..., \mathrm{X}^k\}$ from χ such that (i) $k < n$ and (ii) either χ' is a sub-set of χ or each $\mathrm{X}^i, 1 \leq i \leq k$ is obtained from patterns in χ.

This selection has to be carried out so that there is no significant decrease in classification accuracy. Based on criteria such as classification accuracy, number of prototypes etc., the optimal prototype set has to be obtained. One question that needs to be answered before doing this process is to decide on how many prototypes need to be selected.

3.6.1 Minimal Distance Classifier (MDC)

One simple prototype selection strategy is to use the minimum distance classifier. Each class is represented by the sample mean or centroid of all the samples in the

class. This method selects only one prototype to represent a class. If X_{i1}, X_{i2},..., X_{iN} are the N training patterns for the class i, then the representative sample will be

$$C_i = \frac{\Sigma_{j=1}^{N} X_{ij}}{N}$$

In order to classify a test pattern P, if C_k is the centroid closest to P, it is assigned the class label k of which C_k is the representative pattern.

The major advantage of MDC is that the time complexity is $O(n + mC)$, where $O(n)$ is the time required for computing the centroids and $O(mC)$ is the time required to search for the nearest centroid of the m test patterns. C is the number of classes. The MDC results are the same as the optimal classifier when the classes are normally distributed with a diagonal covariance matrix and the variances in different directions are the same (isotropic classes).

EXAMPLE 15

Consider the example data set given in Figure 3.1. The training set consists of 5 patterns of Class 1, 7 examples of Class 2 and 6 examples of Class 3. The centroid for Class 1 can be obtained by taking the patterns X_1, X_2, X_3, X_4 and X_5 and finding the mean for each coordinate. In this case, the centroid will be (1.0,1.0). Similarly, the centroid for Class 2 will be (4.11, 3) and the centroid for Class 3 will be (3.62, 0.72). This is represented in Figure 3.11.

Consider a test pattern P at (3.0, 2.0). To determine its class, we find the distance of P from the centroids of the three classes.

From P to the centroid of Class 1, the distance is 2.24.

From P to the centroid of Class 2, the distance is 1.49.

From P to the centroid of Class 3, the distance is 1.42.

Since P is closest to the centroid of Class 3, it is classified as belonging to Class 3 according to MDC.

Using MDC, only the distance of P to three points have to be computed whereas using the nearest neighbour classifier, the distance of P to 18 points have to be computed. In general, C distances have to be computed using MDC and n distances computed using the nearest neighbour classifier. It is to be noted that $C << n$ in large-scale applications.

Prototype selection methods can be classified according to the following criteria:

1. Algorithms which select prototypes already available in the training set, i.e., $\chi' \subset \chi$.
2. Algorithms which make use of the training set and produce prototypes which are different from those in the training set, i.e., $\chi' \not\subset \chi$.

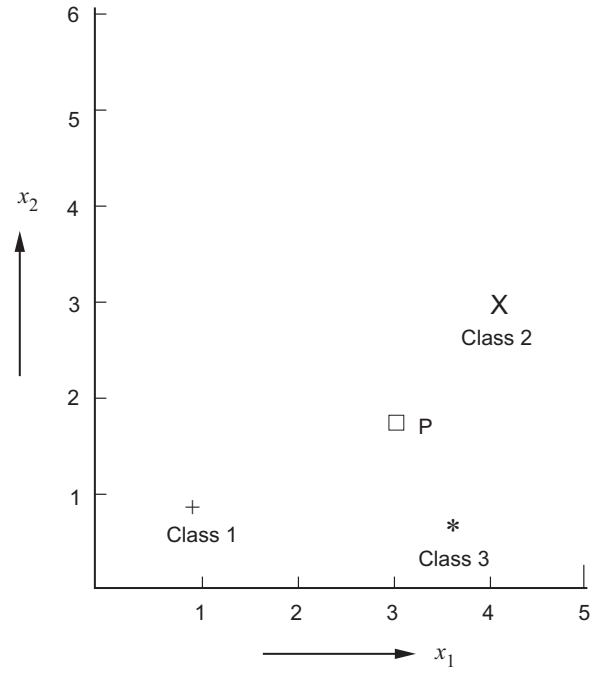


Figure 3.11 Classification using MDC

It should be noted that the second type can be changed to the first type if the nearest sample in the training set is taken as the prototype from the one generated using the algorithm.

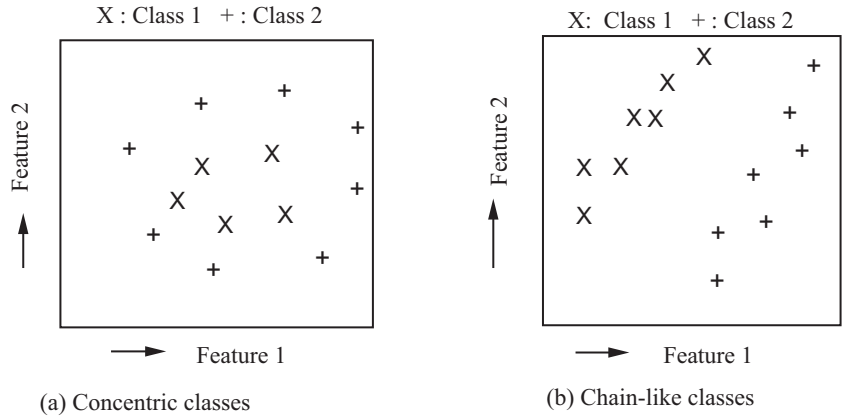
Depending on the type of algorithm used, prototype selection can be basically divided into three types.

1. The first type of algorithm determines a sub-set of the training patterns by classifying the training set itself (or part of it) by using the prototypes selected. These are called condensation algorithms. With these algorithms, we obtain a reduced and consistent set of prototypes without significantly degrading the performance of the original data set.
2. Another type of algorithm involves editing to obtain a sub-set of the data set giving improved performance by removing ''bad'' prototypes.
3. The third type of algorithm clusters the training patterns and uses representative patterns, where one pattern represents each cluster.

In this section, we will discuss the first two methods in detail.

3.6.2 Condensation Algorithms

Even though the MDC is a time-efficient classifier, it fails to give good results when the centroid is not a good representative of the patterns in a class.



(a) Concentric classes (b) Chain-like classes

Figure 3.12 Example of data sets having concentric and chain-like classes

This happens often in nature when the classes are chain-like and elongated in one direction or when two or more classes are concentric which means that patterns of different classes have almost identical sample means. It is illustrated in Figure 3.12. In such cases, it may be meaningful to use more than one representative pattern from each class. One of the first studies on prototype selection is the condensed nearest neighbour (CNN) rule. Other methods include iterative condensation algorithms (ICA), the reduced nearest neighbour (RNN) and modified condensed nearest neighbour (MCNN) algorithms. These methods aim to preserve the integrity of the process by ensuring that the condensed set is consistent with the original set. This means that all the original samples are correctly classified by the condensed set under the nearest neighbour rule. Many of the methods ensure consistency but not minimality of the condensed sub-set. In fact, CNN, which is sensitive to the initial ordering of the training set, can end up with the whole training set (though it is unlikely).

Condensed Nearest Neighbour Algorithm

One of the first and most popularly used methods of prototype selection is the condensed nearest neighbour (CNN) algorithm. In this algorithm, a single pattern is put in the condensed set first. Then each pattern is considered and its nearest neighbour in the condensed set is found. If its label is the same as that of the pattern in the condensed set, it is left out; otherwise the new pattern is included in

the condensed set. After one pass through the training patterns, another iteration is carried out where each training pattern is classified using the condensed set already formed. These iterations are carried out till no more patterns are added to the condensed set. At this stage, the condensed set is consistent. The algorithm is given below.

Let $Train$ be the set of N training pattern pairs given by

$$Train = (\mathrm{X}_1, \theta_1), (\mathrm{X}_2, \theta_2), \cdots, (\mathrm{X}_N, \theta_N).$$

Each pair has a pattern and its class label as the two components. Let $Condensed$ be a set which is empty initially.

STEP 1: Select the first pair from $Train$ and add it to $Condensed$. Let $Reduced$ be the set given by $Train - Condensed$.

STEP 2: Select the first pair from $Reduced$ and find the nearest neighbour of the pattern, in the selected pair, in $Condensed$. If the class label associated with the nearest neighbour and the selected pattern are different, then add the selected pair to $Condensed$. Delete the selected pair from $Reduced$.

STEP 3: Repeat Step 2 till $Reduced$ is empty.

STEP 4: Set $Reduced = Train - Condensed$. Repeat Steps 2 and 3.

STEP 5: Stop if there is no change in $Condensed$ (or $Reduced$) during two successive iterations of Step 4 else iterate through Steps 2, 3 and 4.

EXAMPLE 16

Consider the following set of training samples belonging to three classes, Class 1 represented by a cross, Class 2 represented by a circle and Class 3 represented by a plus.

$\mathrm{X}_1 = (1.0, 1.0, 1)$, $\mathrm{X}_2 = (1.0, 2.0, 1)$, $\mathrm{X}_3 = (1.5, 1.5, 1)$
$\mathrm{X}_4 = (2.0, 2.0, 1)$, $\mathrm{X}_5 = (3.0, 2.0, 2)$, $\mathrm{X}_6 = (4.0, 2.0, 2)$
$\mathrm{X}_7 = (4.0, 3.0, 2)$, $\mathrm{X}_8 = (5.0, 2.5, 2)$, $\mathrm{X}_9 = (2.0, 3.0, 3)$
$\mathrm{X}_{10} = (3.0, 3.5, 3)$, $\mathrm{X}_{11} = (2.0, 4.0, 3)$

This data set is shown in Figure 3.13.

Let the patterns be sent to the algorithm in order from X_1 to X_{11}. First X_1 is put into the set $Condensed$. Since only X_1 is in $Condensed$, X_2, X_3 and X_4 are closest to X_1 and since they have the same class label as X_1, nothing is done. Since X_5 has a class label different from the class label of X_1, it is added to $Condensed$. Now X_6 is compared to X_1 and X_5. Since it is closer to X_5 which has the same label as X_6, nothing is done.

X_7 and X_8 are also closer to X_5 which has the same label, and hence they are not included in *Condensed*. The distance of X_9 from X_1 is 2.236 and the distance of X_9 from X_5 is 1.414. Hence X_9 is closest to X_5 of all the patterns in *Condensed*. But since X_9 has a different label from X_5, it is included in *Condensed*. Now *Condensed* has the patterns X_1, X_5 and X_9. The patterns X_{10} and X_{11} are closest to X_9 in *Condensed* and are therefore not included in *Condensed*. This completes one iteration.

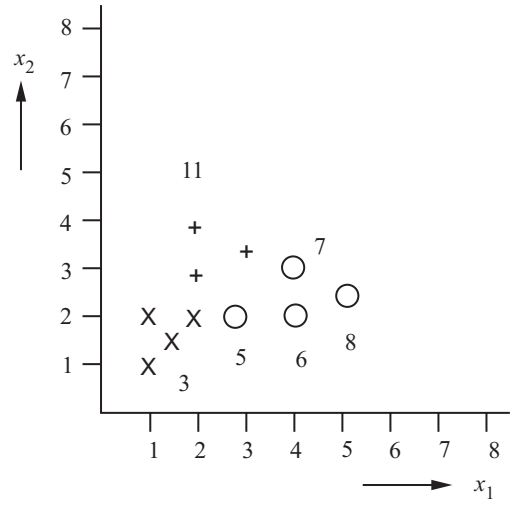


Figure 3.13 Example data set

In the second iteration, X_2 and X_3 are closest to X_1 which has the same label. Therefore, they are not included in *Condensed*. X_4 is equidistant from X_5 and X_9. If X_4 is taken to be closest to X_5 to break the tie, since its label is different from X_5, it is included in *Condensed*. X_6, X_7 and X_8 are closest to X_5 and are not included in *Condensed*. The patterns X_{10} and X_{11} are closer to X_9 which has the same label and therefore are not included in the condensed set.

In the next iteration, since there is no change in *Condensed*, the condensed set consists of the patterns X_1, X_4, X_5 and X_9.

Obtaining the condensed data set is a time-consuming process but once it is available, classification is much faster compared to using the entire training data set. CNN is typically good at reducing the size of the training data set. But this is achieved at the cost of some reduction in the classification accuracy. Also, CNN is *order-dependent*. The set *Condensed* obtained varies in size and contents based on the order in which the training patterns are presented to the above algorithm.

It is very likely that there will be considerable savings in computational expense during the classification phase if this method is used. The trade-off involved is the computational cost of deriving the condensed set, which in itself can be considerable.

But as it involves design time and not the on-line time required for classification, it may not be so significant. This method is sensitive to the initial ordering of the input training data set. Both the contents and the size of the resulting condensed set are likely to change with a different ordering.

EXAMPLE 17

In Figure 3.14, there are six points. The points 1: (1, 2), 2: (2, 2), and 3: (2, 1) belong to one class and the points 4: (2.7, 2), 5: (4, 2) and 6: (4, 1) belong to another class. If the initial ordering of the data is 1, 2, 3, 4, 5 and 6, then the points obtained in the condensed set will be 1, 4 and 2. If the initial ordering is 2, 3, 1, 4, 5 and 6, the condensed set will have 2 and 4. Thus the condensed set is different and of different sizes in the two cases.

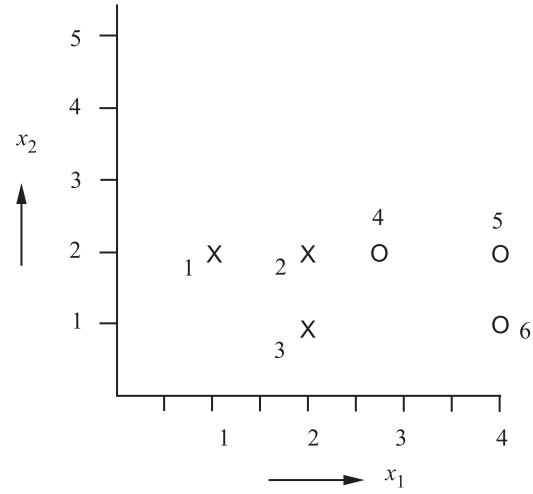


Figure 3.14 Example data set of six points

Modified Condensed Nearest Neighbour Algorithm

In any problem, especially one of high dimensionality, the boundaries of each class are very difficult to determine. The modified condensed nearest neighbour (MCNN) algorithm attempts to partition the region of a class into simpler non-overlapping regions. This is done in an incremental manner, adding prototypes to a representative prototype set till finally all the training patterns are classified correctly using this set of representative prototypes. At this stage, the region pertaining to each class is resolved into approximate Voronoi regions.

$$R_j = \bigcup_{i=1}^{n} V_{ji},\ j = 1, \ldots, c$$

where n is the number of regions in class j. The total number of classes is c.

In this method, a set of prototypes is obtained in an incremental manner. The algorithm starts with a basic set of prototypes comprising one pattern from each class. The training set is classified using these prototypes. Based on the misclassified samples, a representative prototype for each class is determined and added to the set of basic prototypes. Now the training set is classified again with the augmented set of prototypes. Representative prototypes for each class are again determined based on the misclassified samples. This process is repeated till all patterns in the training set are classified correctly. Determining the representative pattern for the misclassified samples in each class is also done in an iterative manner.

The method used for finding a single representative sample of a group of patterns depends on the data set used. One simple method of doing this is to use the centroid as the representative of the group of patterns. Of course, this works well for patterns having a Gaussian distribution with the covariance matrix equal and diagonal. The centroid can also be used if the class can be split up into regions with the above property.

The algorithm for MCNN is given below:

STEP 1: Let the training samples be the set *Train*. Set *Prototype* = ϕ. Set *Typical* = ϕ.

STEP 2: Find a typical pattern for each class from *Train* and put the patterns into *Typical*.

STEP 3: *Prototype* = *Prototype* $\bigcup$ *Typical*

STEP 4: With *Prototype*, classify *Train* using the nearest neighbour algorithm.

STEP 5: Set *Sub-group* = ϕ. Put the misclassified samples into *Sub-group*.

STEP 6: If *Sub-group* = ϕ, stop. *Prototype* gives the set of prototypes finally chosen.

STEP 7: Set *Typical* = ϕ. Find typical patterns for each class in *Sub-group* and put them into *Typical*.

STEP 8: Use *Typical* to classify *Sub-group*.

STEP 9: Find the correctly classified patterns. Set *Sub-group* = ϕ. Put correctly classified patterns in *Sub-group* and if misclassified samples exist, go to Step 7.

STEP 10: Go to Step 3.

The way the typical patterns are found is shown in Steps 5–9. They are found in each class to represent and classify the misclassified patterns. Only the correctly classified samples are kept in *Sub-group*. Typical patterns are found again. This is done till there are no misclassified samples when the typical patterns are used to classify the samples in *Sub-group*.

Using this algorithm, the number of misclassified samples from the training set keeps coming down till all the training patterns are classified correctly by the condensed set (as happens in the case of CNN).

EXAMPLE 18

Consider the example given in Figure 3.15. The centroid C_i is found for each class i. We get

$C_1 = (1.375, 1.625)$

$C_2 = (4.0, 3.0)$

$C_3 = (2.33, 4.0)$

C_1 is closest to X_3, C_2 is closest to X_7 and C_3 is closest to X_{11}.

The prototype set is thus taken as X_3, X_7 and X_{11}. If the training set is classified using this prototype set, all the patterns are correctly classified except X_{10} which is equidistant from X_7 and X_{11}. Since X_7 occurs in the training set before X_{11}, if X_{10} is classified as belonging to Class 2, it is misclassified. The misclassified patterns in each class are used to find the next set of typical patterns which are added to the prototype set. At each iteration, the prototype set keeps building up till all the training patterns are classified correctly.

Unlike the CNN, MCNN is order-independent which means that the algorithm gives the same set of prototypes whatever may be the order of the patterns in the data set. While the design time is higher than that of CNN, the classification accuracy obtained using this method is found to be fairly good.

EXAMPLE 19

In the CNN algorithm, the first pattern in the data set is always put into the condensed set. Then any pattern of the same class as the first pattern is not included into the condensed set. Only when a pattern of another class is encountered, is it put into the condensed set. The patterns which are subsequently put into the condensed set in this iteration and other iterations depend on the patterns already in the condensed set. Therefore, the CNN is an order-dependent algorithm. In the MCNN algorithm, if mean is the typical pattern for a cluster, the mean is found using all the patterns and the closest pattern to the mean is taken as the pattern to be put into the condensed set. This will not depend on the order in which the patterns are presented to the algorithm and makes the MCNN algorithm order-independent. In Figure 3.16, if the closest point to the centroid is taken as the typical sample, then the two patterns included in the prototype set will be 2 and 5. This is true for this data set in whatever order the data is available. The final prototype set will be 2 and 5.

3.6.3 Editing Algorithms

Editing removes samples which are outliers and reduces overlap among classes, thereby giving an improved performance compared to the original data set. Editing algorithms basically estimate the classification error using classification rules, and discards prototypes that are misclassified. The generalised editing scheme can be described as follows:

1. Using the error estimator ϵ, obtain an estimate of the classification error for the rule δ training with the set S. Let R be the set of misclassified samples.
2. Let $S = S - R$.
3. If the terminating condition is satisfied then stop, else go to Step 1.

The performance of the edited prototype set is dependent on the way the editing is done. Some editing methods partition the training set to get two independent sets for designing and testing.

One method of editing finds the k nearest neighbours of every sample and discards the sample if its label does not agree with the label of the largest number of its k neighbours.

The MULTIEDIT is an iterative procedure which uses the principles of ''diffusion'' and ''confusion''. It can be described as follows:

STEP 1: *Diffusion* Make a random partition of the data S into N sub-sets S_1,..., S_N.

STEP 2: *Classification* Classify the samples in S_i using the nearest neighbour rule with $S_{(i+1) \mod N}, i = 1, ..., N$.

STEP 3: *Editing* Discard all the samples that were misclassified.

STEP 4: *Confusion* Pool all the remaining data to constitute the new set S.

STEP 5: *Termination* If Step 3 does not produce any editing, exit with the final solution, else go to Step 1.

EXAMPLE 20

Consider the data set given in Figure 3.15. This consists of the patterns

$X_1 = (1.0, 1.0, 1)$, $X_2 = (1.0, 2.0, 1)$, $X_3 = (1.5, 1.5, 1)$
$X_4 = (2.0, 2.0, 1)$, $X_5 = (1.0, 3.0, 2)$, $X_6 = (3.0, 2.0, 2)$
$X_7 = (4.0, 2.0, 2)$, $X_8 = (5.0, 2.5, 2)$, $X_9 = (3.0, 3.5, 2)$
$X_{10} = (2.0, 3.0, 3)$, $X_{11} = (2.0, 4.0, 3)$, $X_{12} = (3.0, 4.5, 3)$
$X_{13} = (4.0, 3.0, 3)$

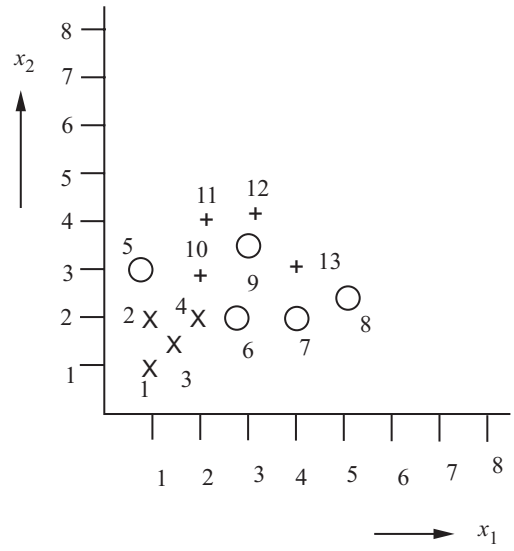


Figure 3.15 Example data set

Each triplet consists of the x-coordinate, y-coordinate and the class label. Let us make three groups S_1, S_2 and S_3 so that

$$S_1 = (X_1, X_2, X_5, X_7, X_{12})$$

$$S_2 = (X_3, X_6, X_8, X_{10})$$

$$S_3 = (X_4, X_9, X_{13}, X_{11})$$

Now if we classify S_1 using S_2 and S_3, the misclassified samples are X_5 and X_{12}. If we classify S_2 using S_1 and S_3, the misclassified sample is X_6. If we classify S_3 using S_1 and S_2, the misclassified samples are X_9 and X_{13}

So the patterns X_5, X_{12}, X_6, X_9 and X_{13} are discarded. This has the effect of removing outliers and making the boundaries of the classes more far apart and towards getting a linear boundary. The remaining patterns are retained giving

$$S = (X_1, X_2, X_3, X_4, X_7, X_8, X_{10}, X_{11})$$

Again, let us make three groups.

$$S_1 = (X_1, X_4, X_{10})$$

$$S_2 = (X_2, X_7, X_{11})$$

$$S_3 = (X_3, X_8)$$

Now we need to repeat the procedure of classifying S_1 using S_2 and S_3, classifying S_2 using S_1 and S_3 and classifying S_3 using S_1 and S_2. The rest of the procedure is left as an exercise.

3.6.4 Clustering Methods

We can obtain representative patterns from a given data set by selecting clusters of high density data regions in multi-dimensional space and replacing each such region by its representative like the centroid of the data in the region. Clustering sub-divides the patterns into groups so that patterns in the same group are similar while patterns belonging to different groups are dissimilar according to some criteria. There are many ways of clustering and they are described in Chapter 9.

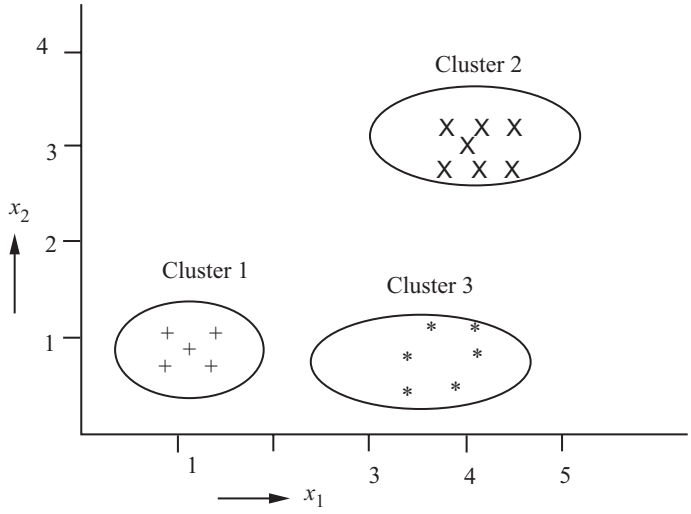


Figure 3.16 Clustering of patterns given in Figure 3.1

EXAMPLE 21

The patterns in Figure 3.1 can be divided into three clusters, one for each class. This is shown in Figure 3.16. If the centroid is the representative pattern of a cluster, Cluster 1 can be represented by the point (1.0,1.0), Cluster 2 can be represented by (4.11, 3) and Cluster 3 can be represented by (3.62, 0.72).

$C_1 = (1.0, 1.0)$

$C_2 = (4.11, 3)$

$C_3 = (3.62, 0.72)$

These centroids can be used as the representative pattern in the class. Now if there is a test pattern P at (4.2,1.8), the distance to the three centroids is

$$d(\mathrm{C}_1, \mathrm{P}) = 3.30$$

$$d(\mathrm{C}_2, \mathrm{P}) = 1.20$$

$$d(\mathrm{C}_3, \mathrm{P}) = 1.23$$

Using the nearest neighbour rule, the pattern P will be classified as belonging to Class 2.

It is also possible to have more clusters for each class. This is shown in Figure 3.17. Each cluster can be represented by one typical pattern.

The centroids of the six clusters are

$$\mathrm{C}_{11} = (1.0, 0.867) \qquad \mathrm{C}_{12} = (1.0, 1.2)$$
$$\mathrm{C}_{21} = (3.8, 3.0) \qquad \mathrm{C}_{22} = (4.24, 3.0)$$
$$\mathrm{C}_{31} = (3.43, 0.65) \qquad \mathrm{C}_{32} = (4.0, 0.85)$$

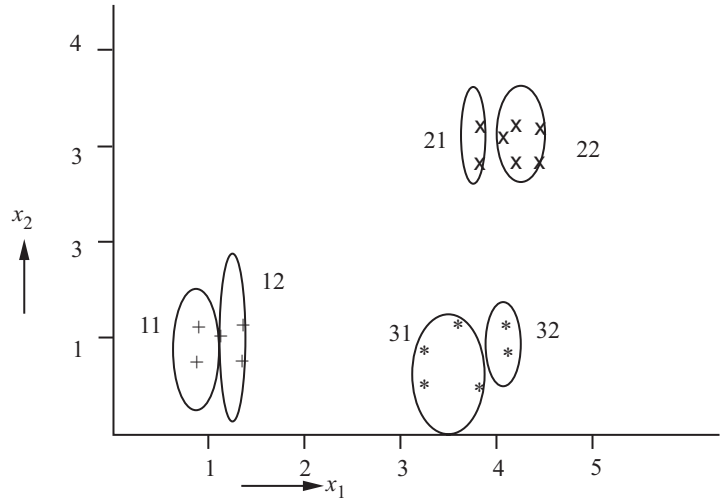


Figure 3.17 Further clustering of the patterns given in Figure 3.1

Taking the distance of the test point P from these centroids, we get

$$d(\mathrm{C}_{11}, \mathrm{P}) = 3.33 \qquad d(\mathrm{C}_{12}, \mathrm{P}) = 3.26$$
$$d(\mathrm{C}_{21}, \mathrm{P}) = 1.26 \qquad d(\mathrm{C}_{22}, \mathrm{P}) = 1.20$$
$$d(\mathrm{C}_{31}, \mathrm{P}) = 1.38 \qquad d(\mathrm{C}_{32}, \mathrm{P}) = 0.97$$

By the nearest neighbour rule, P is closest to the centroid C_{32} and therefore has the same label as the patterns in the cluster which is Class 3.

3.6.5 Other Methods

There are some variations on the CNN in which the algorithms are slightly different or in which different concepts of neighbourhood like the mutual neighbourhood value are used. Some methods use the concept of graph neighbours.

Stochastic techniques can also be applied to the task of prototype selection. The genetic algorithm can be used to find an optimal sub-set of prototypes. In this method, binary chromosomes of length equal to the number of patterns in the original set are used. A ''1'' indicates that the corresponding exemplar is to be included in the sub-set and a ''0'' means that the corresponding exemplar is not included. The classification accuracy obtained by using the reduced set suggested by the chromosome, with a separate validation set is used as the fitness function. A genetic algorithm can be used to do simultaneous editing and feature selection. Here the binary string consists of $n + d$ bits, where n is the number of training patterns and d is the dimensionality of the samples.

Simulated annealing or the Tabu search can be used instead of the genetic algorithm. Here the current solution would be the binary string as described for editing by the genetic algorithm. The evaluation of the strings would also be done in the same way.

Discussion

The nearest neighbour classifiers are intuitively appealing and effective. The drawback is in the implementation since considerable computation time and space will be required for large training data. To reduce the computation time, various approaches have been proposed which result in retrieving the nearest neighbours in a short time. Many of these methods require pre-processing of the training set. This chapter discussed the nearest neighbour classifiers and the various approaches to efficiently find the nearest neighbours to the test patterns.

Prototype selection forms an abstraction of the training set which reduces the time and storage requirement by using the reduced prototype set instead of the complete training set. Various methods for prototype selection were discussed in this chapter.

Further Reading

The nearest neighbour algorithm is described well by Cover and Hart (1967). Patrick and Fischer (1970) proposed the kNN algorithm. The mkNN algorithm is aptly described by Dudani (1976) while the fuzzy kNN algorithm is discussed by Jozwick (1983).

A number of papers have been published on finding efficient algorithms to find the

nearest neighbours of a pattern. Fukunaga and Narendra (1975) explain the branch and bound algorithm to find k nearest neighbours. Miclet and Dabouz (1983)propose an improvement on the branch and bound algorithm. Yunck (1976) identifies the nearest neighbours by projecting the patterns on to a hypercube. Friedman et al. (1975) show how nearest neighbour search can be carried out using projection. Papadimitriou and Bentley (1980) analyse an algorithm for nearest neighbour search using projection. Finding the nearest neighbours using ordered partitions is described by Kim and Park (1986). Broder (1990) describes incremental nearest neighbour search. Other papers on fast algorithms to find nearest neighbours include those by Zhang and Srihari (2004), McNames (2001) and Lai et al. (2007).

The condensed nearest neighbour algorithm which is one of the earliest methods of prototype selection was proposed by Hart (1968). Devi and Murty (2002) describe the modified condensed nearest neighbour algorithm. Dasarathy (1994) found a minimum consistent set (MCS). Gates (1972) describes the reduced nearest neighbour algorithm. The concept of a mutual nearest neighbour is used to find the prototype set by Gowda and Krishna (1979). Construction and use of proximity graphs for finding the prototypes is described by Sanchez (1995). Kuncheva (1995) uses a genetic algorithm to carry out editing of the training patterns. Kuncheva and Jain (1999) use a genetic algorithm to carry out both editing and feature selection simultaneously. Swonger (1972) gives an algorithm for condensation of the training data. Dejiver and Kittler (1980), Tomek (1976) and Wilson (1972) use editing for prototype selection. Chang (1974) and Lam, Keung and Ling (2002; 2002) have also written papers on prototype selection.

Exercises

1. Give an example of a data set for which the kNN classifier gives better results than the NN classifier. Is there an example for which the NN classifier gives better results than the kNN classifier?
2. Give an example where the MkNN gives the correct classification as compared to the kNN classifier.
3. Discuss the performance of the kNN classifier with different values of k. Give an example and show the results obtained as k varies. What happens when $k = n$?
4. Discuss the performance of the MkNN classifier with different values of k. Give an example and show the results obtained as k varies.
5. Show with an example that the CNN algorithm is order-dependent.

6. Give an example where the minimum distance classifier gives good results. Also give an example where the minimum distance classifier gives poor results.

7. Consider the following set of two-dimensional vectors corresponding to classes w_1 and w_2.

w_1	w_2
$(1, 0)$	$(0, 0)$
$(0, 1)$	$(0, 2)$
$(0, -1)$	$(0, -2)$
$(0, -2)$	$(-2, 0)$

   (a) Plot the decision boundary corresponding to the minimum distance classifier.

   (b) Plot the decision boundary corresponding to the nearest neighbour algorithm for classification.

8. Consider the set of two-dimensional patterns :

   (1, 1, 1), (1, 2, 1), (1, 3, 1), (2, 1, 1), (2, 2, 1), (2, 3, 1), (2, 3.5,1), (2.5, 2, 1),(3.5, 1, 1), (3.5, 2, 1), (3.5, 3, 2), (3.5, 4, 2), (4.5, 1, 2), (4.5, 2, 2), (4.5, 3, 2),(5, 4, 2), (5, 5,2),(6, 3, 2), (6, 4, 2), (6, 5, 2)

   where each pattern is represented by feature 1, feature 2 and the class.

   (a) If a test pattern P is at (3.8, 3.1), find the class of P using the nearest neighbour algorithm.

   (b) Find the class of P using the kNN algorithm, where k is 3.

   (c) Find the class of P using MkNN algorithm, where k is 3.

9. Consider the set of two-dimensional patterns:

   (1, 1, 1), (1, 2, 1), (1, 3, 1), (2, 1, 1), (2, 2, 1), (2, 3, 1), (2, 3.5, 1), (2.5, 2, 1),(3.5, 1, 1), (3.5, 2, 1), (3.5, 3,2), (3.5, 4,2), (4.5, 1, 2), (4.5, 2, 2), (4.5, 3, 2), (5, 4, 2), (5, 5, 2), (6, 3, 2),(6, 4,2), (6, 5, 2)

   where each pattern is represented by feature 1, feature 2 and the class. Find the centroid of the two classes. Use the minimum distance classifier to find the class of a point P at $(3.8, 3.1)$.

10. Consider the set of two-dimensional patterns:

    (1, 1, 1), (1, 2, 1), (1, 3, 1), (2, 1, 1), (2, 2, 1), (2, 3, 1), (2, 3.5, 1), (2.5, 2, 1), (3.5, 1, 1), (3.5, 2, 1), (3.5, 3, 2), (3.5, 4, 2), (4.5, 1, 2), (4.5, 2, 2), (4.5, 3, 2), (5, 4, 2), (5, 5, 2), (6, 3, 2), (6, 4, 2), (6, 5, 2)

    where each pattern is represented by feature 1, feature 2 and the class.

    (a) Find the condensed set using condensed nearest neighbour algorithm.

    (b) If the patterns of Class 2 appear first, giving the set
    (3.5, 3, 2), (3.5, 4, 2), (4.5, 1, 2), (4.5, 2, 2), (4.5, 3, 2), (5, 4, 2), (5, 5, 2),(6, 3, 2),(6, 4, 2), (6, 5, 2), (1, 1, 1), (1, 2, 1), (1, 3, 1), (2, 1, 1), (2, 2, 1), (2, 3, 1), (2, 3.5,1), (2.5, 2, 1),(3.5, 1, 1), (3.5, 2, 1)
    find the condensed set using condensed nearest neighbour algorithm.

11. Consider the data set given in Problem 8. Find the condensed set using the modified condensed nearest neighbour algorithm.

Computer Exercises

1. Implement the nearest neighbour algorithm. Use it to classify several test patterns generated randomly for the data set in Figure 3.1.
2. Implement the kNN algorithm. Use it to classify several test patterns generated randomly for the data set in Figure 3.1.
3. Implement the MkNN algorithm. Use it to classify several test patterns generated randomly for the data set in Figure 3.1.
4. Use the NN, kNN and MkNN algorithm to classify several test patterns generated randomly for the data given in Exercise Problems 8, 9 and 10.
5. Use the MDC to classify the data given in Exercise Problems 8, 9 and 10.
6. Implement the CNN algorithm. Use it on the data sets in Exercise Problems 8, 9 and 10. Classify randomly generated test patterns.
7. Choose a large data set divided into training data and test data. Use the training data and classify the test patterns using NN, kNN, and MkNN algorithms. In each case, find the classification accuracy obtained on the test data.
8. Choose a large data set divided into training data and test data. Condense the data set using CNN algorithm. Classify the test patterns using the condensed data and find the classification accuracy.

9. Choose a large data set divided into training data and test data. Condense the data set using MCNN algorithm. Classify the test patterns using the condensed data and find the classification accuracy.

Bibliography

1. Broder, A. J. Strategies for efficient incremental nearest neighbour search. *Pattern Recognition* 23(1/2): 171–178. 1990.
2. Chang, C. L. Finding prototypes for nearest neighbour classifiers. *IEEE Trans. on Computers* C-23(11): 1179–1184. 1974.
3. Cover, T. M. and P. E. Hart. Nearest neighbor pattern classification *IEEE Trans. on Information Theory* IT-13: 21–27. 1967.
4. Dasarathy, Belur V. Minimal consistent set (MCS) identification for optimal nearest neighbour decision system design. *IEEE Trans. on Systems, Man and Cybernetics* 24(3). 1994.
5. Dejiver, P. A. and J. Kittler. On the edited nearest neighbour rule. *Proceedings of the 5th International Conference on Pattern Recognition.* pp. 72–80. 1980.
6. Dudani, S. A. The distance-weighted k nearest neighbor rule. *IEEE Trans. on SMC* SMC-6(4): 325–327. 1976.
7. Friedman, J. H., F. Baskett and L. J. Shustek. An algorithm for finding nearest neighbours. *IEEE Trans on Computers* C-24(10): 1000–1006. 1975.
8. Fukunaga, K. and P. M. Narendra. A branch and bound algorithm for computing k nearest neighbours. *IEEE Trans. on Computers.* pp. 750–753. 1975.
9. Gates, G. W. The reduced nearest neighbour rule. *IEEE Trans. on Information Theory* IT-18(3): 431–433. 1972.
10. Gowda, K.C. and G. Krishna. Edit and error correction using the concept of mutual nearest neighbourhood. *International Conference on Cybernetics and Society.* pp. 222–226. 1979.
11. Hart, P. E. The condensed nearest neighbor rule. *IEEE Trans. on Information Theory* IT-14(3): 515–516. 1968.
12. Jozwik, A. A learning scheme for a fuzzy kNN rule. *Pattern Recognition Letters* 1(5/6): 287–289. 1983.
13. Kim, B. S. and S. B. Park. A fast nearest neighbour finding algorithm based on the ordered partition. *IEEE Trans on PAMI* PAMI-8(6): 761–766. 1986.

14. Kuncheva, L. Editing for the k nearest neighbours rule by a genetic algorithm. *Pattern Recognition Letters* 16(8): 809–814. 1995.

15. Kuncheva, L. and L. C. Jain. Nearest neighbor classifier: Simultaneous editing and feature selection. *Pattern Recognition Letters* 20: 1149–1156. 1999.

16. Lai, Jim Z. C., Yi-Ching Liaw and Julie Liu. Fast k nearest neighbour search based on projection and triangular inequality. *Pattern Recognition* 40: 351–359. 2007.

17. McNames, James. A fast nearest neighbour algorithm based on a principal axis search tree. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 23(9): 964–976. 2001.

18. Miclet, L. and M. Dabouz. Approximative fast nearest neighbour recognition. *Pattern Recognition Letters* 1: 277–285. 1983.

19. Papadimitriou, C. H. and J. L. Bentley. A worst-case analysis of nearest neighbour searching by projection. *Lecture Notes in Computer Science* 85: 470–482. 1980.

20. Patrick, E. A., and F. P. Fischer. A generalized k nearest neighbor rule. *Information and Control* 16: 128–152. 1970.

21. Sanchez, J. S., F. Pla and F. J. Ferri. Prototype selection for the nearest neighbour rule through proximity graphs. *Pattern Recognition Letters* 18(6): 507–513. 1995.

22. Devi, V. Susheela and M. Narasimha Murty. An incremental prototype set building technique. *Pattern Recognition* 35: 505–513. 2002.

23. Swonger, C. W. Sample set condensation for a condensed nearest neighbor decision rule for pattern recognition. *Frontiers of Pattern Recognition.* 511–519. 1972.

24. Tomek, I. A generalization of the kNN rule. *IEEE Trans. on SMC* SMC-6(2): 121–126. 1976.

25. Tomek, I. An experiment with the edited nearest neighbour rule. *IEEE Trans. on SMC* SMC-6(6): 448–452. 1976.

26. Lam, Wai, Chi-Kin Keung and Charles X. Ling. Learning good prototypes for classification using filtering and abstraction of instances. *Pattern Recognition* 35: 1491–1506. 2002.

27. Lam, Wai, Chi-Kin Keung and Danyu Liu. Discovering useful concept prototypes for classification based on filtering and abstraction. *IEEE Trans PAMI* 24(8): 1075–1090. 2002.

28. Wilson, D. L. Asymptotic properties of nearest neighbour rules using edited data. *IEEE Trans. SMC* SMC-2(3): 408–421. 1972.

29. Yunck, Thomas P. A technique to identify nearest neighbours. *IEEE Trans. SMC* SMC-6(10): 678–683. 1976.

30. Zhang, Bin and Sargur N. Srihari. Fast k nearest neighbour classification using cluster-based trees. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 26(4): 525–528. 2004.

4 Bayes Classifier

Learning Objectives

After reading this chapter, you will

- Be able to state the Bayes theorem
- Be able to understand the minimum error rate classifier
- Be able to classify objects using naive Bayes classifiers
- Be able to understand how Bayesian belief networks work

Bayes classifier is popular in pattern recognition because it is an optimal classifier. It is possible to show that the resultant classification minimises the average probability of error. Bayes classifier is based on the assumption that information about classes in the form of prior probabilities and distributions of patterns in the class are known. It employs the posterior probabilities to assign the class label to a test pattern; a pattern is assigned the label of the class that has the maximum posterior probability. The classifier employs Bayes theorem to convert the prior probability into posterior probability based on the pattern to be classified, using the likelihood values. In this chapter, we will introduce some of the important notions associated with the Bayes classifier.

4.1 Bayes Theorem

Let X be the pattern whose class label is unknown. Let H_i be some hypothesis which indicates the class to which X belongs. For example, H_i is the hypothesis that a pattern belongs to class C_i. Let us assume that the prior probability of H_i, given by $P(\mathrm{H}_i)$, is known. In order to classify X, we need to determine $P(\mathrm{H}_i \mid X)$ which is the probability that the hypothesis H_i holds, given the observed pattern X.

$P(\mathrm{H}_i \mid \mathrm{X})$ is the posterior probability of H_i conditioned on X. In contrast, $P(\mathrm{H}_i)$ is the prior probability, or *apriori* probability, of H_i. It is the probability of the hypothesis

regardless of the value of X. The posterior probability, $P(H_i \mid \mathrm{X})$ is based on X and other information (such as background knowledge), whereas the prior probability, $P(\mathrm{H}_i)$ is obtained before observing X.

Similarly, $P(\mathrm{X} \mid \mathrm{H}_i)$ is the probability of X conditioned on H_i. Note that $P(\mathrm{X})$ is defined as a weighted combination of $P(\mathrm{X} \mid \mathrm{H}_i)$ and is

$$P(\mathrm{X}) = \sum_i P(\mathrm{X} \mid \mathrm{H}_i) P(\mathrm{H}_i) \tag{4.1}$$

Bayes theorem is useful in that it provides a way of calculating the posterior probability, $P(\mathrm{H}_i \mid \mathrm{X})$, from $P(\mathrm{H}_i), P(\mathrm{X})$ and $P(\mathrm{X} \mid \mathrm{H}_i)$. It states

$$P(\mathrm{H}_i \mid \mathrm{X}) = \frac{P(\mathrm{X} \mid \mathrm{H}_i) P(\mathrm{H}_i)}{P(\mathrm{X})} \tag{4.2}$$

EXAMPLE 1

In a coffee shop, 99% of the customers prefer coffee. The remaining 1% prefer tea. So, P(coffee drinker) = 0.99 and P(tea drinker) = 0.01. In the absence of any other information, we can classify any customer as a coffee drinker and the probability of error is only 0.01; this is because we are classifying a tea drinker also as a coffee drinker.

We can make a better decision if additional information is available. The following example illustrates this.

EXAMPLE 2

If the prior probability of H that a road is wet is $P(\mathrm{H}) = 0.3$. Then the probability that a road is not wet is 0.7. If we use only this information, then it is good to decide that a road is not wet. The corresponding probability of error is 0.3.

Let us further say that the probability of rain, $P(\mathrm{X})$, is 0.3. Now if it rains, we need to calculate the posterior probability that the roads are wet, i.e., $P(\mathrm{H} \mid \mathrm{X})$. This can be calculated using Bayes theorem. If 90% of the time when the roads are wet, it is because it has rained

$$P(\text{road is wet} \mid \text{it has rained}) = \frac{P(\mathrm{X} \mid \mathrm{H}) \times P(\mathrm{H})}{P(\mathrm{X})} = \frac{0.9 \times 0.3}{0.3} = 0.9$$

Here, the probability of error is 0.1, which is the probability that a road is not wet given that it has rained.

Here, in addition to the prior probability $P(\mathrm{H})$, we require information about $P(\mathrm{X}|\mathrm{H})$ (P(it has rained|roads are wet)) and $P(\mathrm{X})$ (probability of rain).

4.2 Minimum Error Rate Classifier

In general, in pattern classification, we are interested in the average probability of error which corresponds to a weighted probability of errors corresponding to all patterns in a collection, not a single pattern—the weights are the corresponding probabilities of the patterns. The prior probability of a class C is the probability $P(\mathrm{C})$, whereas $P(\mathrm{C}|\mathrm{X})$ is the posterior probability for Class C. Using the training patterns, we can find the posterior probability $P(\mathrm{C}|\mathrm{X})$ for pattern X and Class C. For a test pattern Y, we can find $P(\mathrm{C}|\mathrm{Y})$ for Class C. If the test pattern Y is classified as belonging to Class C, then the probability of error will be $1 - P(\mathrm{C}|\mathrm{Y})$. It can be seen that to reduce the error, a test pattern Y should be classified as belonging to that Class C for which $P(\mathrm{C}|\mathrm{Y})$ is maximum. Further, a minimum error rate classifier can be characterised as follows :

Consider the expected probability of error. It is given by

$$\int_Y (1 - P(\mathrm{C}|\mathrm{Y}))P(\mathrm{Y})d\,\mathrm{Y}$$

For a fixed $P(Y)$, this is minimum when $P(\mathrm{C}|\mathrm{Y})$ is maximum for every Y. If the patterns assume discrete values, then the average probability of error (the expected error) is given by

$$\sum_{\mathrm{Y}_i}(1 - P(\mathrm{C}|\mathrm{Y}_i)P(\mathrm{Y}_i) \tag{4.3}$$

EXAMPLE 3

Let blue, green, and red be three classes with prior probabilities given by

$$P(\text{blue}) = \frac{1}{4} \tag{4.4}$$

$$P(\text{green}) = \frac{1}{2} \tag{4.5}$$

$$P(\text{red}) = \frac{1}{4} \tag{4.6}$$

These three classes correspond to sets of objects coloured blue, green and red respectively. Let there be three types of objects—''pencils'', ''pens'', and ''paper''. Let the class-conditional probabilities of these objects be

$$P(\text{pencil} \mid \text{green}) = \frac{1}{3}; P(\text{pen} \mid \text{green}) = \frac{1}{2}; P(\text{paper} \mid \text{green}) = \frac{1}{6} \tag{4.7}$$

$$P(\text{pencil} \mid \text{blue}) = \frac{1}{2}; P(\text{pen} \mid \text{blue}) = \frac{1}{6}; P(\text{paper} \mid \text{blue}) = \frac{1}{3} \tag{4.8}$$

$$P(\text{pencil} \mid \text{red}) = \frac{1}{6}; P(\text{pen} \mid \text{red}) = \frac{1}{3}; P(\text{paper} \mid \text{red}) = \frac{1}{2} \tag{4.9}$$

Consider a collection of pencil, pen, and paper with equal probabilities. We can decide the corresponding class labels, using Bayes classifier, as follows:

$$P(\text{green} \mid \text{pencil}) = \frac{P(\text{ pencil} \mid \text{green})P(\text{green})}{P(\text{pencil} \mid \text{green})P(\text{green}) + P(\text{pencil} \mid \text{blue})P(\text{blue}) + P(\text{pencil} \mid \text{red})P(\text{red})} \tag{4.10}$$

which is given by

$$P(\text{green} \mid \text{pencil}) = \frac{\frac{1}{3}.\frac{1}{2}}{\frac{1}{3}.\frac{1}{2} + \frac{1}{2}.\frac{1}{4} + \frac{1}{6}.\frac{1}{4}} = \frac{1}{2} \tag{4.11}$$

Similarly, it is possible to compute $P(\text{blue} \mid \text{pencil})$ as

$$P(\text{ blue} \mid \text{ pencil}) = \frac{\frac{1}{2}.\frac{1}{4}}{\frac{1}{3}.\frac{1}{2} + \frac{1}{2}.\frac{1}{4} + \frac{1}{6}.\frac{1}{4}} = \frac{3}{8} \tag{4.12}$$

Finally,

$$P(\text{red} \mid \text{pencil}) = \frac{\frac{1}{6}.\frac{1}{4}}{\frac{1}{3}.\frac{1}{2} + \frac{1}{2}.\frac{1}{4} + \frac{1}{6}.\frac{1}{4}} = \frac{1}{8} \tag{4.13}$$

This would mean that we decide that pencil is a member of class ''green'' because the posterior probability is $\frac{1}{2}$, which is greater than the posterior probabilities of the other classes (''red'' and ''blue''). The posterior probabilities for ''blue'' and ''red'' classes are $\frac{3}{8}$ and $\frac{1}{8}$ respectively. So, the corresponding probability of error, $P(\text{error} \mid \text{pencil}) = \frac{1}{2}$. In a similar manner, for pen, the posterior probabilities are

$$P(\text{green} \mid \text{pen}) = \frac{2}{3}; P(\text{blue} \mid \text{pen}) = \frac{1}{9}; P(\text{red} \mid \text{pen}) = \frac{2}{9} \tag{4.14}$$

This enables us to decide that pen belongs to class ''green'' and $P(\text{error} \mid \text{pen}) = \frac{1}{3}$. Finally, for paper, the posterior probabilities are

$$P(\text{green} \mid \text{paper}) = \frac{2}{7}; P(\text{blue} \mid \text{paper}) = \frac{2}{7}; P(\text{red} \mid \text{paper}) = \frac{3}{7} \tag{4.15}$$

Based on these probabilities, we decide to assign paper to ''red'' which has the maximum posterior probability.

So, $P(\text{error} \mid \text{paper}) = \frac{4}{7}$ and the average probability of error is given by

$$\text{Average probability of error} =$$

$$P(\text{error} \mid \text{pencil}) \times \frac{1}{3} + P(\text{error} \mid \text{pen}) \times \frac{1}{3} + P(\text{error} \mid \text{paper}) \times \frac{1}{3} \tag{4.16}$$

As a consequence, its value is

$$\text{Average probability of error} = \frac{1}{2}.\frac{1}{3} + \frac{1}{3}.\frac{1}{3} + \frac{1}{3}.\frac{4}{7} = \frac{59}{126} \tag{4.17}$$

4.3 Estimation of Probabilities

In the above discussion, we have assumed that the various probabilities like $P(\text{pencil} \mid \text{green})$ are available. However, in a practical scenario, typically we have only the training data of patterns drawn from various classes. We need to estimate these probabilities from the available data. There are several schemes for estimating the probabilities. Two popular schemes are the Bayesian scheme and the maximum likelihood scheme. Here, we assume that the data for each class comes from a known density function with unknown parameter values. The Bayesian scheme is more general and it assumes that the parameters to be estimated are random variables and their prior probabilities are known.

The maximum likelihood scheme is a simple and popularly used scheme for estimating probabilities from the data. Here, the parameters are assumed to be unknown deterministic quantities. We will discuss this scheme in the rest of this section, starting with a simple example to illustrate it.

EXAMPLE 4

Let us assume that there is a classification problem with two classes "C_+" and "C_-". Further, let us assume that the patterns are obtained from these two classes by making a single measurement in the form of x. Let n patterns in the form of $x_1, x_2, x_3, \cdots, x_n$ be the data corresponding to the two classes; out of these n independently drawn patterns, let n_+ be the number from class "C_+" and n_- be the number from class "C_-". Note that $n_+ + n_- = n$. From this data, we can guess the prior probabilities of the classes to be

$$P(C_+) = \frac{n_+}{n} \tag{4.18}$$

$$P(C_-) = \frac{n_-}{n} = 1 - P(C_+) \tag{4.19}$$

The same estimates can be obtained by using the maximum likelihood estimation. Let p and q be the prior probabilities of the classes C_+ and C_-. Then, because the n

patterns are drawn independently, using the binomial distribution, the probability of having n_+ patterns from C_+ out of n patterns is

$$^{n}C_{n_+} p^{n_+} q^{n-n_+} \tag{4.20}$$

This is called the likelihood value, in general, and we can estimate the values of p and q by maximising it. In the process, we can ignore $^{n}C_{n_+}$ because it is a constant with respect to p and q. Further, we know that $q = 1 - p$. So, we need to maximise $p^{n_+}(1-p)^{n-n_+}$. This is maximised when $p = \frac{n_+}{n}$.

The maximum likelihood estimation scheme is used effectively even in the case of continuous random variables where we need to deal with probability density functions. Here, we assume that the form of the underlying probability density function is known and the parameter values are estimated using the maximum likelihood approach. For example, if we assume that n one-dimensional patterns are drawn independently from a class whose distribution is normal or Gaussian, then we can estimate the mean, μ, and variance, σ^2, from the following simple maximum likelihood estimates.

$$\mu = \sum_i \frac{x_i}{n} \tag{4.21}$$

$$\sigma^2 = \frac{1}{n}\sum_i (x_i - \mu)^2 \tag{4.22}$$

Example 5

Let us say that there are two classes C_+ and C_- represented respectively by the collections of patterns {1, 2, 3, 4, 5} and {10, 11, 12, 13, 14}. Then assuming that the two classes are normally distributed, the estimates of the parameters are

$$\mu_+ = \frac{1}{5}(1+2+3+4+5) = \frac{15}{5} = 3; \mu_- = 12 \tag{4.23}$$

$$\sigma_+^2 = \frac{10}{5} = 2; \sigma_-^2 = 2 \tag{4.24}$$

4.4 Comparison with the NNC

When a new pattern has to be classified using the nearest neighbour classifier, it is necessary to find the distance from the test pattern to every pattern in every class and classify it as belonging to the class of the nearest pattern. In the minimum error rate classifier, the posterior probability has to be computed for every class. Usually, the number of classes $C << n$, where n is the number of training patterns and therefore the minimum error rate classifier is much faster. There is no design time for NNC;

whereas for the minimum error rate classifier, it is necessary to calculate the posterior probability for each class using the training data.

From the observed data, the parameters can be estimated using the maximum likelihood estimate. The prior probabilities can be estimated from the data. If the data pertaining to each class is normally distributed, then the data of each class can be represented by the mean and standard deviation. These can be estimated from the data. For class i, if there are N_i points, then the mean will be

$$\mu_i = \frac{1}{N_i} \sum_{j \in C_i} X_j$$

and

$$\sigma_i^2 = \frac{1}{N_i} \sum_{j \in C_i} (X_j - \mu_i)^2$$

and

$$\sigma_i = \sqrt{\frac{1}{N_i} \sum_{j \in C_i} (X_j - \mu_i)^2}$$

The estimated values of the mean and standard deviation represent the set of points in the class.

EXAMPLE 6

Using the data shown in Example 5, we have the estimates as follows. $\mu_+ = 2; \mu_- = 12$ and
$\sigma_+^2 = \sigma_-^2 = 2$. From these, we can obtain the likelihood value for some x to have come from class "+" using the equation

$$\frac{1}{\sqrt{2\pi}\sigma_+} \exp^{-\frac{(x-\mu_+)^2}{\sigma_+^2}} \tag{4.25}$$

We can use these likelihood values and prior probabilities to make decisions. Let us now say, for example, that we have a new pattern whose value of x is 2. The posterior probabilities are given by

$$P(\ \mathrm{C}_+ \mid (x = 2)) = \frac{\text{likelihood of } (x = 2) \text{ for } \mathrm{C}_+ \times P(\ \mathrm{C}_+)}{\text{likelihood of}(x = 2)} \tag{4.26}$$

and

$$P((\ \mathrm{C}_- \mid (x = 2)) = \frac{\text{likelihood of } (x = 2) \text{ for } \mathrm{C}_- \times P(\ \mathrm{C}_-)}{\text{likelihood of}(x = 2)} \tag{4.27}$$

Note that we have the same denominator in both the right-hand side expressions. So, we can decide to assign the pattern ''2'' to the class with the larger numerator. Assuming equal prior probabilities of $\frac{1}{2}$ to the classes C_+ and C_-, we need to use only the likelihood values while making decisions. Note that the likelihood values for C_+ and C_- for $x = 2$ respectively are $\frac{1}{2\sqrt{\pi}}$ and $\frac{1}{2\sqrt{\pi}} \exp^{-50}$. So, we assign the pattern $x = 2$ to class C_+.

4.5 Naive Bayes Classifier

A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes theorem where every feature is assumed to be class-conditionally independent.

4.5.1 Classification using Naive Bayes Classifier

Classification of objects where a large number of features and classes exist becomes difficult since it would require an enormous number of observations to estimate the probabilities.

Naive Bayes classifiers assume that the effect of a variable value on a given class is independent of the values of other variables. This assumption is called class-conditional independence. It is made to simplify the computation and in this sense, it is considered to be naive.

The assumption is fairly strong and is sometimes not applicable. But studies comparing classification algorithms have found the naive Bayesian classifier to be comparable in performance with classification trees and neural network classifiers. They have also exhibited high accuracy and speed when applied to large databases.

4.5.2 The Naive Bayes Probabilistic Model

Abstractly, the probability model for a classifier is a conditional model

$$p(C|F_1, \ldots, F_n)$$

over a dependent class variable C with a small number of outcomes or classes, conditional on several feature variables F_1 through F_n. The problem is that if the number of features is large or when a feature can take on a large number of values, then working with such a model using probability tables is not feasible. We therefore reformulate the model to make it more tractable.

Using Bayes theorem, we can write

$$p(C|F_1, \ldots, F_n) = \frac{p(C)\ p(F_1, \ldots, F_n|C)}{p(F_1, \ldots, F_n)}.$$

The denominator does not depend on C and the values of the features F_i are given, so the denominator is effectively constant. The numerator is equivalent to the joint probability model

$$p(C, F_1, \ldots, F_n)$$

which can be rewritten as follows, using repeated applications of the definition of conditional probability:

$$\begin{aligned}
&p(C, F_1, \ldots, F_n) \\
&\quad = p(C)\ p(F_1, \ldots, F_n|C) \\
&\quad = p(C)\ p(F_1|C)\ p(F_2, \ldots, F_n|C, F_1) \\
&\quad = p(C)\ p(F_1|C)\ p(F_2|C, F_1)\ p(F_3, \ldots, F_n|C, F_1, F_2) \\
&\quad = p(C)\ p(F_1|C)\ p(F_2|C, F_1)\ p(F_3|C, F_1, F_2)\ p(F_4, \ldots, F_n|C, F_1, F_2, F_3)
\end{aligned}$$

and so forth. Using the naive conditional independence assumptions, each feature F_i is conditionally independent of every other feature F_j for $j \neq i$. This means that

$$p(F_i|C, F_j) = p(F_i|C)$$

and so the joint model can be expressed as

$$\begin{aligned}
p(C, F_1, \ldots, F_n) &= p(C)\ p(F_1|C)\ p(F_2|C)\ p(F_3|C)\ \cdots p(F_n|C) \\
&= p(C) \prod_{i=1}^{n} p(F_i|C)
\end{aligned}$$

This means that under the above assumptions of independence, the conditional distribution over the class variable C can be expressed as

$$p(C|F_1, \ldots, F_n) = \frac{1}{Z} p(C) \prod_{i=1}^{n} p(F_i|C)$$

where Z is a scaling factor dependent only on $F_1, \ldots, F_n$, i.e., a constant if the values of the feature variables are known.

Models of this form are much more manageable, since they use prior probabilities of classes $p(C)$ and independent probability distributions $p(F_i|C)$. If there are k classes and if a model for $p(F_i)$ can be expressed in terms of r parameters, then the corresponding naive Bayes model has $(k-1) + nrk$ parameters. In practice, often $k = 2$ (binary classification) and $r = 1$ (Bernoulli variables as features) are common, and so the total number of parameters of the naive Bayes model is $2n + 1$, where n is the number of binary features used for prediction.

4.5.3 Parameter Estimation

All model parameters (i.e., class priors and feature probability distributions) can be computed from the training set. These are maximum likelihood estimates of the probabilities. If a given class and feature value never occur together in the training set then the frequency-based probability estimate will be zero. This will make some probabilities zero when they are multiplied with this probability. It is therefore necessary to incorporate a small-sample correction in all probability estimates so that no probability is ever set to be exactly zero.

The range of values of a feature in a class compared to the range over all the classes will be used to calculate the feature probability distribution. For instance, consider a feature which takes only the values 0 and 1. In a class, the ratio of the number of patterns having the feature value as 0 to the total number of patterns in the class, gives the probability of the feature being 0 in this class.

EXAMPLE 7

The training set contains examples belonging to different classes. The ratio of the number of examples of a particular class and the total number of examples gives the class prior for that particular class. In this way, the class priors can be calculated for every class.

Consider a training set as follows:

Total number of examples = 100

Number of examples of Class 1 = 40

Number of examples of Class 2 = 30

Number of examples of Class 3 = 30

Therefore,

$$\text{Prior probability of Class 1} = \frac{40}{100} = 0.4$$

$$\text{Prior probability of Class 2} = \frac{30}{100} = 0.3$$

$$\text{Prior probability of Class 3} = \frac{30}{100} = 0.3$$

Out of the 40 examples of Class 1, if a binary feature takes 0 in 30 examples and 1 in 10 examples, then the prior probability that this feature is 0 in this class will be $\frac{30}{40} = 0.75$.

4.5.4 Constructing a Classifier from the Probability Model

The naive Bayes classifier combines the Bayes probability model with a decision rule. One common rule is to pick the hypothesis that is most probable; this is known as the maximum a posterior or MAP decision rule. The corresponding classifier is the function ''classify'' defined as follows:

$$\text{classify}(f_1, \ldots, f_n) = \underset{c}{\text{argmax}}\ p(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c)$$

Despite the fact that the far-reaching independence assumptions are often inaccurate, the decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one-dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality, such as the need for data sets that scale exponentially with the number of features. Like all probabilistic classifiers under the MAP decision rule, it arrives at the correct classification as long as the correct class is more probable than any other class; hence class probabilities do not have to be estimated very well. In other words, the overall classifier is robust enough to ignore serious deficiencies in its underlying naive probability model.

EXAMPLE 8

Consider the example given in Chapter 6, Table 6.3. Suppose only the first ten patterns are there in the data set. This data set is shown in Table 4.1. Consider a new pattern

Table 4.1 Example training data set

Cook	Mood	Cuisine	Tasty
Sita	Bad	Indian	Yes
Sita	Good	Continental	Yes
Asha	Bad	Indian	No
Asha	Good	Indian	Yes
Usha	Bad	Indian	Yes
Usha	Bad	Continental	No
Asha	Bad	Continental	No
Asha	Good	Continental	Yes
Usha	Good	Indian	Yes
Usha	Good	Continental	No

Cook = Sita, Mood = Bad, Cuisine = Continental

We need to classify this example as Tasty = yes or Tasty = no. Since there are 6 examples out of 10 with Tasty = yes, the prior probability P(Tasty = yes) is $\frac{6}{10}$, which is 0.60. The prior probability P(Tasty = no) is $\frac{4}{10}$ which is 0.40. There are 2 examples with Cook = Sita and Tasty = yes and 0 examples with Cook = Sita and Tasty = no. The probability for Cook = Sita given that Tasty = yes will be

$$P(\text{Cook} = \text{Sita} \mid \text{Tasty} = \text{yes}) = \frac{2}{6} = 0.33$$

$$P(\text{Cook} = \text{Sita} \mid \text{Tasty} = \text{no}) = 0$$

The probability is zero, and since it is multiplied by other probabilities, those probabilities will also be zero. To avoid this, a small value is taken. Let it be 0.01.

There are 2 examples with Mood = Bad and Tasty = yes and 3 examples with Mood = Bad and Tasty = no. Therefore

$$P(\text{Mood} = \text{Bad} \mid \text{Tasty} = \text{yes}) = \frac{2}{6} = 0.33$$

$$P(\text{Mood} = \text{Bad} \mid \text{Tasty} = \text{no}) = \frac{3}{4} = 0.75$$

There are 2 examples with Cuisine = Continental and Tasty = yes and 3 examples with Cuisine = Continental and Tasty = no. Therefore

$$P(\text{Cuisine} = \text{Continental} \mid \text{Tasty} = \text{yes}) = \frac{2}{6} = 0.33$$

$$P(\text{Cuisine} = \text{Continental} \mid \text{Tasty} = \text{no}) = \frac{3}{4} = 0.75$$

Therefore,

$$P(\text{Tasty} = \text{yes} \mid \text{X}) = 0.6 \times 0.33 \times 0.33 \times 0.33 = 0.0216$$

$$P(\text{Tasty} = \text{no} \mid \text{X}) = 0.4 \times 0.01 \times 0.75 \times 0.75 = 0.00225$$

The new pattern is therefore classified as belonging to the class Tasty = yes as $P(\text{Tasty} = \text{yes} \mid \text{X}) > P(\text{Tasty} = \text{no} \mid \text{X})$.

4.6 Bayesian Belief Network

A Bayesian network (or a belief network) is a probabilistic graphical model that represents a set of variables and their probabilistic dependencies. Formally, Bayesian networks are directed acyclic graphs whose nodes represent variables, and whose arcs encode conditional dependencies between the variables. There are efficient algorithms that perform inference and learning in Bayesian networks. Bayesian networks that model sequences of variables (for example speech signals or protein sequences) are called dynamic Bayesian networks. Generalisations of Bayesian networks that

can represent and solve decision problems under uncertainty are called influence diagrams.

If there is an arc from node A to another node B, A is called a parent of B, and B is a child of A. The set of parent nodes of a node X_i is denoted by its parents (X_i). A directed acyclic graph is a Bayesian network relative to a set of variables if the joint distribution of the node values can be written as the product of the local distributions of each node and its parents:

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i | \text{ parents}(X_i))$$

If node X_i has no parents, its local probability distribution is said to be unconditional, otherwise it is conditional. If the value of a node is observed, then the node is said to be an evidence node.

Each variable is associated with a conditional probability table which gives the probability of this variable for different values of its parent nodes. The value of

$$P(X_1, ..., X_n)$$

for a particular set of values for the variables can be easily computed from the conditional probability table.

EXAMPLE 9

Ram is a student. He loves going to the movies. He will go to the theatre in the evening if he has money in his pocket. On the other hand, if it rains, he will not go the theatre. When Ram does not go to the movies, he stays home and watches the television. He also dedicates some time to his studies.

Figure 4.1 shows the belief network for this situation. The variables involved are

1. Money in Ram's pocket (M)
2. Rain (R)
3. Ram goes to the movies (G)
4. Ram studies (S)
5. Ram watches television (T)

Each variable has a conditional probability table. The variables M and R are not influenced by any factor here, so the conditional probability table for these variables has only one value, i.e., $P(M)$ and $P(R)$ respectively. The conditional probability table for G shows the probability of G, given values of M and R, i.e., $P(G|M$ and $R)$. The

variables S and T are influenced by G. The conditional probability table for S gives $P(S|G)$ and the conditional probability table for T gives $P(T|G)$.

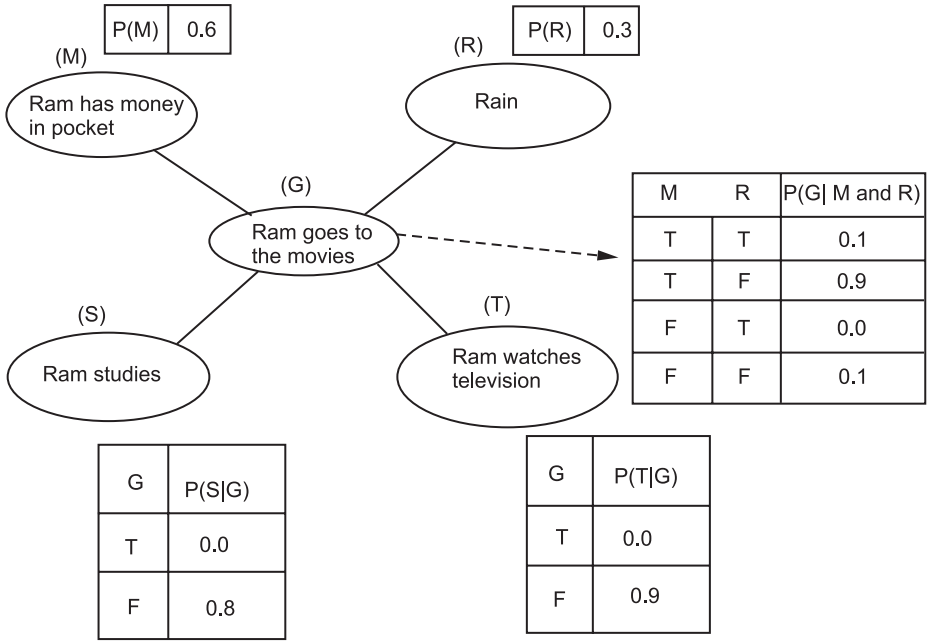


Figure 4.1 Belief network of how Ram spends an evening

Let $\neg A$ stand for negation of the proposition A. Now if we want to know the probability that it does not rain, Ram does not have money, Ram goes to the movies and Ram does not watch television, we want $P(\neg R, \neg M, G, \neg T)$. This will be

$$P(\neg R) \times P(\neg M) \times P(G \mid \neg R \text{ and } \neg M) \times P(\neg T \mid G)$$

These values can be directly obtained from the conditional probabilities tables. So

$$P(\neg R, \neg M, G, \neg T) = 0.7 \times 0.4 \times 0.1 \times 1.0 = 0.028$$

It can be seen that the probability of any combination of variables like $P(X_1, X_2, ...)$ can be obtained from the belief network using the conditional probability tables.

Discussion

This chapter introduces the reader to the classification of patterns based on the probability of it belonging to a class given the data. The minimum error rate classifier classifies a pattern as belonging to the class in which it has the highest posterior probability. The naive Bayes classifier assumes that all the features are independent given the class and computes $P(F \mid C)$ as a product of probabilities of the form $P(F_i \mid C)$. An important property of the naive Bayes classifier is its simplicity in dealing with probability estimates. It can successfully estimate relevant probabilities using a smaller number of training patterns. The belief network aids in finding probabilities which can be used for classification.

Further Reading

The book by Duda et al. book (2001) on pattern classification is a well-known and excellent resource material on Bayes decision theory and Bayes classifiers. It also covers some of the other relevant topics like Bayesian and maximum likelihood estimation schemes. Another interesting is the book by Russell and Norvig (2003) on artificial intelligence. It has an excellent coverage of probabilistic reasoning including a very good discussion on Bayesian networks. Domingos and Pazzani (1997) have explained in detail why the naive Bayes classifier works. Another researcher who has carried out interesting work in this area is Irina Rish. Rish's paper on naive Bayes classifier (2001) is very informative.The book by Pearl (1988) is an early and classic resource for a good discussion on probabilistic networks.

Heckerman (1999) gives an excellent introduction to learning Bayesian networks and presents efficient learning schemes. Neapolitan (2003) has written a book which deals with learning Bayesian networks. The excellent book by Tan et al.(2007) on data mining covers material on naive Bayes classifier in a simple and critical manner. The chapter on Bayesian techniques in the book by Bishop (2003)'s book deals with various important issues associated with Bayesian learning.

Exercises

1. Consider Example 3 discussed in Section 4.2. Compute the posterior probabilities for classes ''green'', ''blue'' and ''red'' for pen and paper. Obtain $P(\text{error} \mid \text{pen})$ and $P(\text{error} \mid \text{paper})$.
2. Show that the maximum value of $p^{n_+}(1-p)^{n-n_+}$ is obtained when $p = \frac{n_+}{n}$. [HINT: Maximise the logarithm of the function instead of the function itself as logarithm is a monotonic function].

3. Consider a coin with a prior probability of 0.5 for heads and 0.5 for tails. If the coin is tossed four times and gives heads, heads, tails and heads respectively, compute the probability of occurence of heads and tails.

4. Consider the maximum likelihood estimates for mean and variance for the normal density given in equations (4.21) and (4.22). Using these, obtain the estimated values for means and variances from the data given below for the two classes specified.
   Class 1 : 1, 2, 3, 4
   Class 2 : 7, 8, 9

5. Consider the following data set :

Feature 1	Feature 2	Feature 3	Class
0	0	0	0
1	0	1	1
1	0	0	0
1	1	1	1
0	1	1	1
0	1	1	0

   If we then have a test pattern with feature 1 as 0, feature 2 as 0 and feature 3 as 1, classify this pattern using NNC and Bayes classifier.

6. Consider the Bayesian belief network given in Figure 4.1. Consider another variable ''has exams''. How would you include it in the belief network? Put down the likely values for the conditional probability tables for all the variables.

7. Classify the test pattern in Example 2 using naive Bayes classifier.

8. The profit that a woman selling flowers makes depends on how fresh the flowers are. Besides, if there is a festival approaching, her profit increases. On the other hand, towards the end of the month, most people do not buy flowers. If the woman makes enough profit, she cooks a good feast for dinner.

   Draw the belief network for the above facts and give the likely conditional probability tables for all the variables. Using the information in the conditional probability tables, find the probability that the woman cooks a good feast given that the flowers are fresh.

Computer Exercises

1. Consider two classes with distributions $N(0, 1)$ and $N(5, 1)$. Generate 5 patterns from each of these classes and estimate the means and variances of the two

normally distributed classes using these patterns with the maximum likelihood estimation scheme. Increase the number of patterns from 5 to 50 for each class. Estimate the means and variances. What can you infer? What happens when the number of patterns per class is increased to 500?

2. Write a program to obtain the posterior probabilities from the classes described in Exercise 1 above with equal prior probabilities. Now consider patterns in the range 0 to 10 for classification. How are they classified?

3. Write a program to perform classification using the naive Bayes classifier. Generate a three-dimensional data set and run the classifier. Observe its behaviour as the dimensionality of the data set increases. Increase each class to 50 and estimate the means and variances.

Bibliography

1. Duda, R. O., P. E. Hart, and D. G. Stork. *Pattern Classification.* Second Edition. Wiley-Interscience. 2001.

2. Russell, S. and P. Norvig. *Artificial Intelligence: A Modern Approach.* Pearson India. 2003.

3. Domingos, P. and M. Pazzani On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning* 29: 103–130. 1997.

4. Pearl, J. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kauffman. 1988.

5. Rish, I. An empirical study of the naive Bayes classifier. *IJCAI Workshop on Empirical Methods in Artificial Intelligence.* 2001.

6. Heckerman, D. Bayesian networks for knowledge discovery. In *Advances in Knowledge Discovery and Data Mining* edited by U. M. Fayyad, G. P. Shapiro, P. Smyth, and R. Uthurusamy. MIT Press. 1996.

7. Tan, P. N., M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Pearson India. 2007.

8. Neapolitan, R. E. *Learning Bayesian Networks.* Upper Saddle River, NJ: Prentice Hall. 2003.

9. Bishop, C. M. *Neural Networks for Pattern Recognition.* New Delhi: Oxford University Press. 2003.

5 Hidden Markov Models

Learning Objectives

At the end of this chapter, you will:

1. Have an understanding of Markov models: You will be able to understand the parameters associated with Markov models and be able to use them in computing the associated probabilities
2. Learn to use hidden Markov models (HMMs) to solve three problems:
   (a) Calculating the probability of occurrence of a particular observation sequence
   (b) Finding the input sequence which maximises the joint probability of the observation sequence and the state sequence
   (c) Finding the HMM parameters to maximise the probability of the observation sequence
3. Understand how to use Markov models for classification

Hidden Markov models (HMMs) are important in pattern recognition because they are ideally suited to classify patterns where each pattern is made up of a sequence of sub-patterns. For example, assume that a day is either *sunny*, *cloudy*, or *rainy* corresponding to three different types of weather conditions. Then a typical week during summer could be described as sunny, sunny, sunny, sunny, sunny, sunny, sunny corresponding to every day of the week being sunny. Similarly, it is possible that every day in a week during the rainy season can be rainy for which the week can be characterised as rainy, rainy, rainy, rainy, rainy, rainy, rainy.

In this example, we assumed that a week is a pattern and is made up of seven days or sub-patterns. Even though ideally a week in summer is expected to have seven consecutive sunny days, in reality, there could be variations. For example, some days during summer could possibly be cloudy or rainy.

Let us use S to denote a sunny day, C for a cloudy, and R for a rainy day. Then,

in general, each week is a sequence of seven symbols where each of the symbols is either an S, a C, or an R. Equivalently, we can say that a week starts with a symbol (corresponding to the first day), and goes through a sequence of six more symbols; here, for any two successive days, we have a symbol for the first day and a transition to the second symbol. Such a transition from a state to another is probabilistic. For example, transition from S to S is more likely during summer. Similarly, transition from S to R is more probable during the rainy season.

HMMs can be used to characterise classes of patterns, where each pattern is viewed as a sequence of states. It is possible that the actual state is hidden and only its probabilistic variations are observed. For example, we may be locked inside a room and not be in a position to see whether it is rainy, sunny, or cloudy. But, we may be able to observe whether a visitor is carrying an umbrella or not and hear the sound of rain hitting the ground or the roof of the house. Based on these observations, we learn the HMM associated with each class and use these models in classification.

Conventionally, HMMs are popular in speech and speaker recognition tasks. Here, a speech signal is viewed as a sequence of phonemes—different speech utterances will have different sequences of phonemes/symbols. There are several other important applications including recognition of protein and DNA sequences/sub-sequences in bioinformatics. We will introduce some of the important notions associated with HMMs in this chapter.

5.1 Markov Models for Classification

In a Markov model, we start with a state and go through a sequence of states based on transitions that are characterised in a probabilistic manner. Each state is directly observed. Even though there are several applications to demonstrate the underlying concept, we start with a simple coin tossing scenario that is popularly used to illustrate the idea. Let us say that there are two different biased coins; further, let the first coin, say coin1, be biased towards $heads$ and the second coin, say coin2, be biased towards $tails$. Let us use the symbol h for heads and t for tails. Now suppose we toss each of these coins separately and observe the sequence of outcomes which, after 10 tosses in each case, are given by

coin1: hhhthhhhth and coin2: ttthtthttt

Further, let one of the coins be picked randomly and tossed six times. Let the sequence obtained be hhhhth, then is it possible to predict whether the coin picked is coin1 or coin2? This is a two-class classification problem.

Simply by observing the sequence, hhhhth, it may be possible to conclude that the coin used is coin1 because there is a bias towards heads. However, note that it

is possible that this sequence is generated by tossing coin2 also, but with a lower probability. We illustrate this using Example 1.

EXAMPLE 1

Consider the data described in Table 5.1.

Table 5.1 Probabilities of each coin showing heads or tails

Coin considered	Probability of heads	Probability of tails
coin1	0.9	0.1
coin2	0.1	0.9

Let us consider the probability of generating the sequence, hhhhth, by tossing coin1 six times independently. The first toss gives h with a probability of 0.9. Similarly, second, third, and fourth tosses also produce h with a probability of 0.9 in each case. In the fifth independent toss of coin1, we get t with a probability of 0.1 and finally the sixth toss gives us h with a probability of 0.9. So, the probability of generating the sequence hhhhth using six independent tosses of coin1 is

$$P_1(\text{hhhhth}) = (0.9) \times (0.9) \times (0.9) \times (0.9) \times (0.1) \times (0.9) = 0.059049 \approx 0.06 \quad (5.1)$$

Similarly, let us consider the probability of generating the sequence hhhhth by tossing coin2 six times independently. The first toss gives h with a probability of 0.1. Second, third, and fourth tosses also produce h with a probability of 0.1 in each case. In the fifth independent toss of coin2, we get t with a probability of 0.9 and finally the sixth toss produces h with a probability of 0.1. So, the probability of generating the sequence hhhhth using six independent tosses of coin2 is

$$P_2(\text{hhhhth}) = (0.1) \times (0.1) \times (0.1) \times (0.1) \times (0.9) \times (0.1) = 0.000009 \approx 0.00001 \quad (5.2)$$

The above two probabilities indicate that even though there is a non-zero probability of generating the sequence hhhhth by using either coin1 or coin2, it is approximately 6000 times more likely for coin1 to have generated the sequence than coin2. We consider additional examples of this variety in Exercise 1 at the end of the chapter.

Let us consider another example to illustrate the idea further. In Example 2, we consider a toy data set of printed characters of size 3×3. In reality, either printed or handwritten characters could be bigger in size (e.g., 64×64 or 128×128 or even 200×100) based on the signal processing system used on the scanned images of characters.

EXAMPLE 2

Consider two printed characters each of size 3×3 as shown in Table 5.2.

Note that the character on the left is ''1'' (observe column 3 which has a collection of 1s) and the character on the right is ''7'' (observe the first row and the third column which have only 1s and no 0s). We can analyse these patterns as follows. In the 3×3 pattern corresponding to digit 1, the first row is ''001''; let us label this state as S_1. The remaining two rows, that is row 2 and row 3, are the same as the first row. So, the states corresponding to digit 1 are S_1, S_1, and S_1, considering each row as a state. Similarly, analysing the digit 7 shown on the right side, the first row is ''111'' followed by ''001'' and ''001'' as the second and third rows. Labelling 111 as S_2, we can see that the sequence of states in this case is S_2, S_1, S_1.

Table 5.2 Printed characters 1 and 7

0 0 1	1 1 1
0 0 1	0 0 1
0 0 1	0 0 1

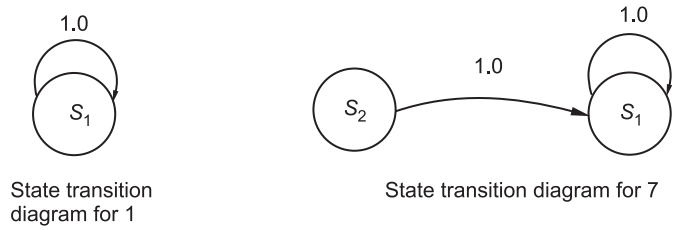


Figure 5.1 State transition for the digits 1 and 7

The corresponding state transition diagrams are shown in Figure 5.1. Note that such simple state transition diagrams emerge because of the nature of the classification problem. In our example, each digit is represented as a small size matrix of 3×3 binary pixel values. In the case of digit 1, the sequence of the three states is S_1, S_1, S_1. While, in the case of 7, the sequence is S_2, S_1, S_1. Here, each digit has three rows and each state is a row. It is possible to view the transitions based on the columns, where each column corresponds to a state. This is discussed in Exercise 2.

We can solve a variety of problems using this simple model. In order to attempt it, we need to introduce some assumptions.

1. *First-order Markov assumption* Let the transitions be taking place at time instances $t_1, t_2, \cdots$. Let the state at t_i be denoted by S_{t_i}. Consider that at t_1, the state is i and at time t_2, there is a transition to state j. Now, let us estimate the probability of transition to k at t_3 given by

$$P(S_{t_3} = k | S_{t_2} = j, S_{t_1} = i) = P(S_{t_3} = k | S_{t_2} = j) \tag{5.3}$$

We made a simplifying assumption here, a first-order Markov assumption, which states that the state at time t_n (S_{t_n}), for any legal (non-negative) value of $n-1$, only depends on the state at t_{n-1} ($S_{t_{n-1}}$). This means that in equation (5.3), once we know the state at t_2, we do not require state at t_1 to predict the state at t_3.

2. *Stationarity assumption* In general, transition from one state to another depends on the time instance at which the transition takes place. This means that we need a separate transition probability matrix for each time instance. In order to reduce the overheads, the following stationarity assumption is used which means that the probabilities characterising the state transitions do not vary with time. So, to characterise the model, we need a single state transition matrix that does not vary with time.

$$P(S_{t_n} = i | S_{t_{n-1}} = j) = P(S_{t_{n+l}} = i | S_{t_{n+l-1}} = j), l \geq 0 \tag{5.4}$$

We consider another example keeping the Markovian and stationarity assumptions in mind. Specifically, the example involves four sports channels on television (TV).

EXAMPLE 3

Consider the problem of selecting a sports channel on TV from four channels: ESPN, DD Sports, Star Sports, and Zee Sports. Assume that each selected channel is viewed for a fixed period of time, say one hour. Further, there is a probabilistic transition from a channel to another channel as specified below. In general, transition from one state to another depends on the time instance at which the transition is taking place. For example, transition from Zee Sports to DD Sports may be more likely at 4 PM on a week day than that at 6 AM. However, for the sake of simplicity, we assume that the transition probabilities do not vary with time.

Consider that at t_1, the channel selected is Zee Sports (Z) and at time t_2, there is a transition to DD Sports (D). We need to estimate the probability of transition to Z at t_3. Let us consider some details associated with the problem.

- *States (Channels)*
  1. ESPN (E)
  2. DD Sports (D)
  3. Star Sports (S)
  4. Zee Sports (Z)

- *State transition matrix* As noted earlier, we assume the stationarity property and as a consequence, a single state transition probability matrix is adequate. It is shown in Table 5.3. Note that the entries in each row adds up to 1; this is because in a transition, we move from the current state to one of the states in the collection.

Table 5.3 Probabilities of transition from one channel to another

TV channel	ESPN	DD Sports	Star Sports	Zee Sports
ESPN	0.4	0.3	0.2	0.1
DD Sports	0.2	0.5	0.2	0.1
Star Sports	0.1	0.2	0.6	0.1
Zee Sports	0.1	0.1	0.1	0.7

Now, let us estimate the probability of transition to Z at t_3 given that at t_1, the channel is Z and at t_2 it is D, using the equation

$$P(S_{t_3} = Z|S_{t_2} = D, S_{t_1} = Z) = P(S_{t_3} = Z|S_{t_2} = D) \tag{5.5}$$

Note that we are able to obtain the right-hand side of the equation using the first-order Markovian assumption. We can read the probability in the right-hand side of equation (5.5) from Table 5.3—the second row (the row for D) and the fourth column (for Z) which is 0.1.

We can solve a variety of problems using this simple model. In this example, we have effectively used the first-order Markovian property to simplify the computation as shown in equation (5.5). We illustrate it further using Examples 4 and 5.

Example 4

Using the data provided in Example 3, given that at the current time instance, the state is Z, what is the probability that the state at the next time instance is D and at the subsequent time instance, it is Z?

We can estimate it as follows:

$$P(S_{t_3} = Z, S_{t_2} = D|S_{t_1} = Z) = P(S_{t_3} = Z|S_{t_2} = D, S_{t_1} = Z) \times P(S_{t_2} = D|S_{t_1} = Z) \tag{5.6}$$

$$= P(S_{t_3} = Z|S_{t_2} = D) \times P(S_{t_2} = D|S_{t_1} = Z) \tag{5.7}$$

$$= (0.1) \times (0.1) = 0.01 \tag{5.8}$$

Note that equation (5.6) is based on Bayes theorem and conditional probability. The first term in the right-hand side of equation (5.7) is based on the first-order Markovian assumption. The entries in equation (5.8) are obtained from the matrix given in Table 5.3.

Example 5

Consider the data given in Example 3. Let the channel selected at t_i be D. What is the probability that at t_{i+2}, the channel selected is D?

Note that the state at t_{i+1} is left unspecified here. So, it could be E, D, S, or Z. As a consequence, the probability to be obtained is

$$\begin{aligned}
P(S_{t_{i+2}} = D|S_{t_i} = D) &= P(S_{t_{i+2}} = D, S_{t_{i+1}} = E|S_{t_i} = D) + P(S_{t_{i+2}} = D, S_{t_{i+1}} = D|S_{t_i} = D) \\
&\quad + P(S_{t_{i+2}} = D, S_{t_{i+1}} = S|S_{t_i} = D) + P(S_{t_{i+2}} = D, S_{t_{i+1}} = Z|S_{t_i} = D) \\
&= P(S_{t_{i+2}} = D|S_{t_{i+1}} = E) \times P(S_{t_{i+1}} = E|S_{t_i} = D) + \\
&\quad P(S_{t_{i+2}} = D|S_{t_{i+1}} = D) \times P(S_{t_{i+1}} = D|S_{t_i} = D) + \\
&\quad P(S_{t_{i+2}} = D|S_{t_{i+1}} = S) \times P(S_{t_{i+1}} = S|S_{t_i} = D) + \\
&\quad P(S_{t_{i+2}} = D|S_{t_{i+1}} = Z) \times P(S_{t_{i+1}} = Z|S_{t_i} = D) \\
&= (0.3)(0.2) + (0.5)(0.5) + (0.2)(0.2) + (0.1)(0.1) \\
&= 0.06 + 0.25 + 0.04 + 0.01 = 0.36
\end{aligned}$$

We can use Markov models in pattern classification. We assume that the start state and state transition probabilities are adequate to use the Markov models effectively. Then the training phase consists of using the training patterns as sequences of states and learning the state transition probabilities. For each class, we may get different matrices. For example, consider the digits ''1'' and ''7'' described in Example 2. Recall that digit 1 is described by the state sequence S_1, S_1, S_1 and digit 7 is characterised by the sequence S_2, S_1, S_1. The corresponding state transition diagram is given in Figure 5.1.

Example 6

We can use the state transition probabilities to classify a test pattern. Consider, for example, a test pattern shown in Table 5.4. The corresponding state sequence is S_1, S_1, S_1 and so it is classified as digit 1.

Table 5.4 Test pattern to be classified as either digit 1 or digit 7

0	0	1
0	0	1
0	0	1

The test pattern provided and the scheme for classification shown in Example 6 clearly bring out the shortcomings of the simple model. It also shows the rigidity of the description of the classes in even such a simple representation of digits of size 3×3 pixels. A more realistic classification scenario requires to be more general and noise tolerant. In order to illustrate it, we describe a noisy version of digit 1 in Example 7.

EXAMPLE 7

Consider the noisy version of digit 1 of size 3×3 shown in Table 5.5. The pattern in Table 5.5 may be represented, row wise, as a sequence S_1, S_1, S_4, where the third row ''1 0 1'' is represented by state S_4. Assuming that digit 1 starts with state S_1 and digit 7 with S_2, the test pattern is closer to 1 and is correctly classified.

Table 5.5 Noisy version of digit 1 as a test pattern

0	0	1
0	0	1
1	0	1

However, we will have difficulty if there is noise in the first row itself as shown in Table 5.6. Here, the digit corresponds to the sequence S_4, S_1, S_1 and based on the first state which is S_4, it matches neither with digit 1 nor with digit 7.

Table 5.6 Noisy version of digit with noise in the first row

1	0	1
0	0	1
0	0	1

One way to deal with noise is to increase the number of states and define the classes using the enhanced set of states. For example, in addition to the description of digit 1 using the sequence S_1, S_1, S_1, we can also use the sequence S_1, S_1, S_4 to describe a noisy version of digit 1 shown in Table 5.5. However, the digit shown in Table 5.6 is equally likely to be a ''1'' or a ''7''; by replacing the entry 1 in the first row and first column by 0, we get digit 1 and by replacing the entry 0 in the first row and second column by 1, we get digit 7. Further, this scheme of increasing the number of states and remembering all the noisy and non-noisy versions of digits will lead to over-fitting when used in classification.

So, we update the model by including a better probabilistic structure to deal with

noisy digits by viewing the observations as noisy instantiations of the state sequences. In such a model, the states may not be directly observable. They are hidden; so, the model is called a *hidden Markov model*. We discuss it in detail in the next section.

5.2 Hidden Markov Models

In hidden Markov models (HMMs), the states are not observed; they are hidden. State transitions are probabilistic and observations are probabilistic functions of state; this will take care of noisy versions of the states without increasing their number. As a consequence, the classifier will avoid overfitting and generalise well. We start with some simple examples before we formally introduce the model in detail.

EXAMPLE 8

Consider two coins, coin1 and coin2, which are tossed and the outcome is disclosed. One of the two coins is selected randomly and tossed. The outcome (h or t) is disclosed. However, the information on which coin is selected for tossing is not disclosed. For example, the observation sequence is ''hhthhtthhhttt...''. In order to abstract the generation process, we need to know which coin (state) is selected, the probability of generating ''h'' or ''t'' based on the selected coin (or the bias of the coin), and how the coin selection sequence (or the state transition sequence) is characterised. Specifically, we need the following information:

- The first coin to have been selected (start state)
- The state transition probability matrix (this gives us the probability of a coin selection sequence)
- The probability of generating an observation (h or t) given that a specific coin is selected for tossing

This is the most frequently used example to illustrate HMMs. Here, the states are hidden, because we do not know which coin is used to generate the current symbol (observation). We look at another example.

EXAMPLE 9

Consider the details of the four sports channels provided in Example 3. Let us say that the TV is located in the hall and the person in the study room can only make out whether the programme currently being watched in the hall is a cricket, football, or a tennis match. The channel information is hidden from the person in the study room; only the programme is known. This is another example where the generation process can be characterised by an HMM. As in Example 8, the states (channels) are hidden

but observations (programmes) are known. We need to use the observation sequence to learn the state transition probabilities, probability for a state to be the start state and the probability of an observation (programme), given the state (channel).

EXAMPLE 10

We consider the example of digit recognition which is more relevant to classification. This is a modified version of the problem specified in Example 2. We also consider noisy patterns. The training examples of digit ''1'' are shown in Table 5.7 and training examples for digit ''7'' in Table 5.8.

Table 5.7 Training data for digit "1" including noisy versions

0	0	1	1	0	1	0	1	1	0	0	0
0	0	1	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	0	0	1
0	0	1	1	0	1	0	1	1	0	0	0
0	0	1	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	1	1	0	0	0

Table 5.8 Training data for digit "7" including noisy versions

1	1	1	0	1	1	1	0	1	1	1	0
0	0	1	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	0	0	1
1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	1	0	1	0	1	1	0	0	0
0	0	1	0	0	1	0	0	1	0	0	1
1	1	1	1	1	1	1	1	1	1	1	1
0	0	1	0	0	1	0	0	1	0	0	1
0	0	1	1	0	1	0	1	1	0	0	0

Let us consider a noisy variant of a state as a possible observation. For example, ''1 0 1'' may be viewed as a noisy version of the state S_1 (that is ''0 0 1''). In this simple case of 3×3 digits, we permit at most one bit to be affected by noise. So, in this case

also the states are hidden and observations are viewed as noisy versions of the hidden state. The training data of digits can be viewed as observation sequences and it is used to learn the HMM. In most classification problems including digit recognition, the sequence of states involved for each class is known in advance. This will be exploited in the learning process.

5.2.1 HMM Parameters

Before we get into details, we will describe the various elements of the hidden Markov model which are:

- N = number of states (coins; TV channels; rows of the character matrix) in the model
- M = number of distinct observation symbols (''h'' or ''t'' ($M = 2$); 3 programmes per sports channel)
- L = length of the observation sequence; the size of the training data set in terms of the number of states

Using the training data for each class, we compute the corresponding HMM specified by $\lambda = (A, B, \pi)$, where

- the matrix $A = [a_{ij}]; a_{ij} = P(S_{t+1} = j | S_t = i)$, which stands for the probability of transition from state i to state j at time t
- the observation probability $b_j(k) = P(O_t = k | S_t = j), 1 \leq k \leq M$, is the probability of observing symbol k at time t ($O_t = k$) when the state is j
- the initial state probability $\pi_i = P(S_{t_1} = i), 1 \leq i \leq N$, the probability of starting in state i

We assume that for each class under consideration, we are given a training data set of observations $\mathcal{O} = (O_1, O_2, \cdots, O_L)$. We use these observations to learn λ. Some of the related problems are:

1. Estimate the HMM $\lambda = (A, B, \pi)$ that maximises $P(\mathcal{O}|\lambda)$
2. Given $\mathcal{O}$ and λ, find the optimal state sequence $\mathcal{S} = (S_{t_1}, S_{t_2}, \cdots, S_{t_L})$
3. Given λ, compute $P(\mathcal{O}|\lambda)$, the probability of occurrence of $\mathcal{O}$ given the model

5.2.2 Learning HMMs

In the simple classification problem considered in Example 10, we have two classes corresponding to the digits ''1'' and ''7''. We have 12 training patterns for each of the classes. Digit 1 is characterised by the state sequence S_1, S_1, S_1 and digit ''7'' is described by the sequence S_2, S_1, S_1. So, in this example, there are only two states:

- S_1 : 0 0 1
- S_2 : 1 1 1

We consider each row of each of the 12 training patterns corresponding to a class and learn the parameters as follows. We start with the patterns of digit ''1''.

1. Consider each row of a pattern as an observation symbol; here each row is a 3 bit binary pattern. There are a total of 36 observation symbols/rows—there are 12 patterns and each pattern has 3 rows. There are two states ''0 0 1'' and ''1 1 1''. Let there be two clusters corresponding to these two states; cluster 1 is represented by ''0 0 1'' (S_1) and cluster 2 by ''1 1 1'' (S_2).
2. We cluster the 36 observations into the two clusters based on the distance between the cluster representative and the observation considered. We use the Hamming distance which can be defined as the number of mismatching bits. For example, consider the first row of the first pattern which is ''0 0 1''. It is assigned to cluster 1 because the distance between this observation and ''0 0 1'' (S_1) is 0 whereas the distance between ''0 0 1'' and ''1 1 1'' (S_2) is 2 because 2 out of 3 bits mismatch.
3. Out of the 36 observations (rows), 27 are ''0 0 1''; they are assigned to cluster 1 because they are closer to S_1 (''0 0 1'') than to S_2 (''1 1 1''). There are 3 observations that are ''0 0 0''; these are also assigned to cluster 1 as the distance between S_1 and ''0 0 0'' is 1 whereas the distance between S_2 and ''0 0 0'' is 2. So, 30 observations are assigned to cluster 1.
4. The remaining 6 observations may be assigned to either cluster 1 or cluster 2 as S_1 or S_2 are equally distant (distance is 1) from these observations. From here onwards in this chapter, we assume that they are assigned to cluster 2; in other words, whenever there is a tie, we uniformly decide in favour of cluster 2.
5. We calculate the various HMM parameters as follows:

   (a) $\pi_i = \dfrac{\text{Number of times the first observation is assigned to cluster } i}{\text{Total number of patterns}}, \; i = 1, 2$

   Specifically, estimates of π_is are

   $$\pi_1 = \frac{10}{12} = \frac{5}{6} \text{ and } \pi_2 = \frac{2}{12} = \frac{1}{6}$$

   (b) $a_{ij} = \dfrac{\text{Number of times } O_t \text{ is in cluster } i \text{ and } O_{t+1} \text{ is in cluster } j}{\text{Number of times} O_t \text{ is in cluster } i}$ for any t

   The estimates of a_{ij}s are

   $$a_{11} = \frac{16}{20} = \frac{4}{5} \text{ and } a_{12} = \frac{4}{20} = \frac{1}{5}$$

   $$a_{21} = \frac{4}{4} = 1 \text{ and } a_{22} = \frac{0}{4} = 0$$

(c) There are different ways of estimating $b_j(k)$, the probability of generating observation symbol k at time t when the state is j. We compute the probabilities based on the distance of the observation symbol from each of the states or cluster representatives, in this case from S_1 and S_2. The specific algorithm is as follows:

STEP 1: Compute the distance between the observation vector k and each of the N states. We use the Hamming distance. Let the distances be $d_1, d_2, \cdots, d_N$. Then

$$b_j(k) = 1 - \frac{d_j}{\text{maximum distance possible}}$$

STEP 2: Normalise the distances so that $\sum_j b_j(k) = 1$. This is obtained by dividing each value by the sum of the non-normalised values. Use the normalised probabilities.

Let us compute $b_1(001)$ and $b_2(001)$ with respect to the data given in Table 5.7. The states are S_1 = 0 0 1 and S_2 = 1 1 1. The distances are $d_1 = 0$ and $d_2 = 2$. Note that the maximum possible distance = 3. The values before normalisation are:

$b_1(001) = 1 - \frac{0}{3} = 1$ and $b_2(001) = 1 - \frac{2}{3} = \frac{1}{3}$.

Sum of these values is $\frac{4}{3}$. So, the values after normalisation are:

$b_1(001) = \frac{1}{\frac{4}{3}} = \frac{3}{4}$ and $b_2(001) = \frac{\frac{1}{3}}{\frac{4}{3}} = \frac{1}{4}$.

In a similar manner, we can compute the probabilities associated with the other observation symbols: ''1 0 1'', ''0 1 1'', and ''0 0 0''. The normalised values are:

$b_1(101) = \frac{1}{2}$ and $b_2(101) = \frac{1}{2}$

$b_1(011) = \frac{1}{2}$ and $b_2(011) = \frac{1}{2}$

$b_1(000) = 1$ and $b_2(000) = 0$

(d) Similarly, from the data given in Table 5.8 in Example 10, we can estimate the parameter values of the HMM corresponding to digit 7 (Class 2):

$\pi_1 = \frac{0}{12} = 0$ and $\pi_2 = \frac{12}{12} = 1$

$a_{11} = \frac{4}{5}$, $a_{12} = \frac{1}{5}$, $a_{21} = \frac{6}{7}$, and $a_{22} = \frac{1}{7}$

There are seven different observation symbols (rows) in the training patterns given in Table 5.8. They are ''1 1 1'', ''0 0 1'', ''0 1 1'', ''1 0 1'', ''1 1 0'', and ''0 0 0''. The corresponding normalised b values are:

$b_1(111) = \frac{1}{4}$ and $b_2(111) = \frac{3}{4}$

$b_1(001) = \frac{3}{4}$ and $b_2(001) = \frac{1}{4}$

$b_1(011) = \frac{1}{2}$ and $b_2(011) = \frac{1}{2}$
$b_1(101) = \frac{1}{2}$ and $b_2(101) = \frac{1}{2}$
$b_1(110) = 0$, and $b_2(110) = 1$
$b_1(000) = 1$, and $b_2(000) = 0$

Note that $b_i(k)$ depends on the observation symbol k and the state i; it does not depend on the digit here. So, we get the same values of $b_i(k)$ for either class 1 (digit 1) or class 2 (digit 7) using the above algorithm for computing the probabilities.

5.3 Classification Using HMMs

In the previous section, we discussed how to train HMMs. We specifically obtained the HMMs associated with the digits ''1'' and ''7'' in a two-class classification problem. We summarise in Table 5.9, the models λ_1 and λ_7 associated with digits ''1'' and ''7'' respectively.

Table 5.9 Parameters of the HMMs of two classes: digit 1 and digit 7

HMM for digit 1 (λ_1)	HMM for digit 7 (λ_7)
$\pi_1 = \frac{5}{6}$, $\pi_2 = \frac{1}{6}$	$\pi_1 = 0$, $\pi_2 = 1$
$a_{11} = \frac{4}{5}$, $a_{12} = \frac{1}{5}$, $a_{21} = 1$, $a_{22} = 0$	$a_{11} = \frac{4}{5}$, $a_{12} = \frac{1}{5}$,$a_{21} = \frac{6}{7}$,$a_{22} = \frac{1}{7}$
$b_1(001) = \frac{3}{4}$, $b_2(001) = \frac{1}{4}$	$b_1(111) = \frac{1}{4}$, $b_2(111) = \frac{3}{4}$
$b_1(101) = \frac{1}{2}$, $b_2(101) = \frac{1}{2}$	$b_1(011) = \frac{1}{2}$, $b_2(011) = \frac{1}{2}$
$b_1(000) = 1$, $b_2(000) = 0$	$b_1(110) = 0$, $b_2(110) = 1$
$b_1(010) = \frac{1}{2}$, $b_2(010) = \frac{1}{2}$	$b_1(100) = \frac{1}{2}$, $b_2(100) = \frac{1}{2}$

Note that we have the same $b_i(k)$ values for both the digits as mentioned earlier for all legal values of i and k. So, we have shown them across the columns of digit 1 and digit 7 without writing them separately under each column.

5.3.1 Classification of Test Patterns

We use the models characterised by the parameters shown in Table 5.9 to classify the test patterns. We start with a simple example.

EXAMPLE 11

Consider the test pattern, $test_1$, shown in Table 5.4 for classification. We compute the probability for λ_1 to have generated $test_1$ and for λ_7 to have generated $test_1$; we assign $test_1$ to that class which has a higher probability of generating it. More specifically, we have

$$P(test_1|\lambda) = \pi_1 \times b_1(001) \times a_{11} \times b_1(001) \times a_{11} \times b_1(001)$$

Note that we are computing the conditional probability as follows. The first row of $test_1$ is closer to S_1 than to S_2. So, it is in cluster 1 and the start state is S_1 which has a probability of π_1. The symbol generated is ''0 0 1'' in state 1 which has a probability given by $b_1(001)$. The second row in $test_1$ is ''0 0 1'' which is in cluster 1 represented by S_1. So, the transition from the first row to the second row is characterised by the probability a_{11} as both the rows are assigned to cluster 1. Again the probability of generating the observation symbol ''0 0 1'' from S_1 is $b_1(001)$. In a similar manner, transition in this example from row 2 to row 3 is characterised again by a_{11} and the corresponding probability of generating ''0 0 1'' in the third row from S_1 has a probability of $b_1(001)$. The product of all these probabilities gives us $P(test_1|\lambda)$. We plug in the values of various parameters from the HMM corresponding to digit 1 (λ_1) given in Table 5.9 to compute $P(test_1|\lambda_1)$. This is given by

$$P(test_1|\lambda_1) = \frac{5}{6} \times \frac{3}{4} \times \frac{4}{5} \times \frac{3}{4} \times \frac{4}{5} \times \frac{3}{4} = \frac{9}{40} = 0.225$$

In a similar manner, we can compute $P(test_1|\lambda_7)$ by plugging in the values corresponding to the parameters of HMM for digit 7 from Table 5.9 and it is given by

$$P(test_1|\lambda_7) = 0 \times \frac{3}{4} \times \frac{4}{5} \times \frac{3}{4} \times \frac{4}{5} \times \frac{3}{4} = 0$$

Here, $P(test_1|\lambda_1)(= 0.225)$ is greater than $P(test_1|\lambda_7)$ (= 0). So, $test_1$ is classified as digit 1 as it has a higher probability of being generated by λ_1 than that of λ_7. We consider a noisy example test pattern to see how the models can be used to classify it in the next example.

EXAMPLE 12

Let us consider the pattern, $test_2$, shown in Table 5.5 for classification. The first row of the pattern is ''0 0 1'' which is assigned to cluster 1 represented by S_1; the second row ''0 0 1'' is assigned to cluster 1 represented by S_1; and the third row ''1 0 1'' is assigned to cluster 2 represented by S_2. We calculate the probabilities for $test_2$ to have been generated by the two HMMs as follows:

$$P(test_2|\lambda) = \pi_1 \times b_1(001) \times a_{11} \times b_1(001) \times a_{12} \times b_2(101)$$

This is similar to that of $test_1$ but for the last row which is ''1 0 1'' and is assigned to cluster 2. So, the transition from row 2 to row 3 is characterised by a_{12} and $b_2(101)$ gives the probability of generating row 3 (''1 0 1'') from S_2. The corresponding entries for λ_1 and λ_7 are obtained by plugging in the corresponding values from Table 5.9 as shown below:

$$P(test_2|\lambda_1) = \frac{5}{6} \times \frac{3}{4} \times \frac{4}{5} \times \frac{3}{4} \times \frac{1}{5} \times \frac{1}{2} = \frac{3}{80} = 0.0375$$

In a similar manner, we can compute $P(test_2|\lambda_7)$ by plugging in the values corresponding to the parameters of HMM for digit 7 and it is given by

$$P(test_2|\lambda_7) = 0 \times \frac{3}{4} \times \frac{4}{5} \times \frac{3}{4} \times \frac{1}{5} \times \frac{1}{2} = 0$$

Here again we have $P(test_2|\lambda_1)$ is greater than $P(test_2|\lambda_7)$. So, we classify $test_2$ as digit 1. We consider one more noisy example next to illustrate the ideas further.

EXAMPLE 13

Let us consider the test pattern, $test_3$, shown in Table 5.6. Here the first row ''1 0 1'' is assigned to cluster 2 because of a tie. The second row ''0 0 1'' and the third row ''0 0 1'' are assigned to cluster 1. So, the corresponding probabilities are computed as follows:

$$P(test_3|\lambda) = \pi_2 \times b_2(101) \times a_{21} \times b_1(001) \times a_{11} \times b_1(001)$$

Correspondingly, for λ_1 we have

$$P(test_3|\lambda_1) = \frac{1}{6} \times \frac{1}{2} \times 1 \times \frac{3}{4} \times \frac{4}{5} \times \frac{3}{4} = \frac{3}{80} = 0.0375$$

and for λ_7, we have

$$P(test_3|\lambda_7) = 1 \times \frac{1}{2} \times \frac{6}{7} \times \frac{3}{4} \times \frac{4}{5} \times \frac{3}{4} = \frac{27}{140} = 0.193$$

In this case, $P(test_3|\lambda_7)$ is greater than $P(test_3|\lambda_1)$. So, we classify $test_3$ as digit 7. Note that the first row ''1 0 1'' is equidistant from S_1 and S_2 and we have resolved ties in favour of S_2 in case of a tie. That is why $test_3$ is assigned to digit 7. The classification when the tie is resolved in favour of S_1 is considered in an exercise at the end of the chapter.

Discussion

This chapter introduced the reader to classification of patterns using hidden Markov models. Specifically, patterns were considered as sequences of observations. Further,

the states were hidden and had to be obtained from the observations in the form of training data. HMMs and how they are used in classification were described. In order to explain HMMs, we started with a brief introduction to Markov models which assume that the current state is characterised only by the immediate previous state (first-order Markovian property). Further, we assumed the stationarity property to simplify the models. We discussed, using several simple examples, how HMMs are used in classification. We considered the digit recognition problem and worked out the details. We assumed that each row of the digit is an observation; the observation may be a noisy version of the hidden state. We provided a scheme for learning the HMMs from the training data and showed how to use the learned models in classification.

Further Reading

Rabiner's book on hidden Markov models (1989) is the first and most authoritative coverage on HMMs.

A number of tutorials are available on HMMs. Dugad and Desai (1996) give an excellent coverage of HMM. Fosler-Lussier's book (1998) provides highly accessible material on both Markov models and HMMs. Kanungo (1998) has written lecture notes on HMMs in the form of slides which give enough details for a beginner. It also gives several simple examples.

Two books which cover HMMs are the following. Duda et al.'s book (2001) is a good resource for discussion on HMMs. It also covers some of the practically relevant examples, while the book written by Rabiner and Juang (1993) provide a comprehensive coverage of HMMs with applications to speech signal processing.

Exercises

1. Consider the data given in Table 5.1 and Example 1. Obtain the probabilities for
   (a) generating sequence hhhthhhhth using coin1 and coin2.
   (b) generating sequence ttthtthttt using coin1 and coin2.
2. Consider the digit data described in Table 5.2. Assuming that each column corresponds to a state, obtain the state transition diagrams for the digits 1 and 7 described as 3×3 matrices.
3. Draw the state transition diagram corresponding to the transition probabilities shown between various channels in Table 5.3.
4. Using the data in Table 5.3, obtain the probability that at t_4, the state is D, given that at t_1, t_2, and t_3, the states are E, S, and Z respectively.

5. Solve Exercise 4 if the states are D, D, and D at t_1, t_2, and t_3 respectively.
6. Consider the data given in Table 5.3. Obtain the probability that at t_2 and t_3, the states are S and E respectively, given that at t_1, the channel selected is Z.
7. If at t_1 the state is S, then obtain the probability that the state at t_3 is Z using the data given in Example 3.
8. Consider the data given in Table 5.10. There are two digits 4 (on the left side) and 9 (on the right side), each of size 3×3. Draw the corresponding state transition diagrams. You may label the states (rows) appropriately. For example, you may label the row ''0 0 1'' as S_1, row ''1 1 1'' as S_2, and row ''1 0 0'' as S_3.

Table 5.10 Printed characters 4 and 9

1 0 0	1 1 1
1 1 1	1 1 1
0 0 1	0 0 1

9. Observe that there is a one-to-one correspondence between the state transition diagram and the state transition matrix. Obtain the state transition matrices for the digits 4 and 9 shown in Table 5.10.
10. Consider the 3×3 binary matrix of digits ''1'' and ''7'' shown in Table 5.7 and Table 5.8. If each row is a state, then how many observation symbols (perhaps noisy versions) are possible per state? You may assume that a noisy version of a state can have a change of at most one bit in the state sequence. For example, ''0 1 1'' is a noisy variant of S_1, but ''1 1 1'' is not. How about the total number of noisy digits of size 3×3?
11. Consider Table 5.7 in Example 10 and the discussion in Section 5.2.2. Identify the 6 observations that are equidistant from either S_1 or S_2.
12. Consider the data given in Example 11. Work out the non-normalised values of $b_1(k)$ and $b_2(k)$, for k = 1 0 1, k = 0 1 1, and k = 0 0 0. Obtain the normalised values from the non-normalised values and verify the correctness of the values given in item 5(c) of Section 5.2.2 for the digit 1.
13. Compute the values of the parameters of the HMM for digit 7 given in item 5(d) of Section 5.2.2 and verify the correctness of the values provided.

14. Consider the parameter values given in Table 5.9. Re-estimate the values of various parameters if the decision is to resolve the ties in assigning observations to clusters as follows. In case of digit 1, assign the observation vector to S_2 and in case of digit 7, assign it to S_1 if there is a tie, which means that the observation symbol is equidistant from S_1 and S_2.

15. Consider the test pattern given in Table 5.11. Use the data given in Table 5.9 to classify it.

Table 5.11 Test pattern to be classified as either digit 1 or digit 7

1	1	1
0	0	1
0	0	1

16. Consider classification of $test_3$ where the first row ''1 0 1'' is equidistant from S_1 and S_2. Assign it to cluster 1 represented by S_1. How do you classify $test_3$ using the parameter values provided in Table 5.11?

Computer Exercises

1. Consider a two class problem where one class is digit 1 and the other is digit 7. Assume that each digit is a binary matrix of size 40×30; there are 40 rows and 30 columns. Use the state transition diagrams shown in Figure 5.1 to simulate a training data set of 1000 1s and 1000 7s. Assume that S_1 is a 30 bit sequence with 29 0s and a 1; and S_2 is a bit sequence of length 30 with all the bits as 1. Add noise randomly by mutating up to 100 entries in each digit. Here, mutation means replacing ''0'' by ''1'' at a randomly selected location in the digit of size 40×30.

2. Write a program to estimate the parameters of the HMMs corresponding to digit 1 and digit 7 from a random collection of 900 patterns picked from each digit (class) generated in the above exercise.

3. Write a program to classify the remaining 200 patterns generated in Computer Exercise 1 using the models obtained in Computer Exercise 2.

Bibliography

1. Dugad, Rakesh and U. B. Desai. *A Tutorial on Hidden Markov Models.* Technical Report No. SPANN-96.1. IIT Bombay, Mumbai. 1996.
2. Fosler-Lussier, Eric. *Markov Models and Hidden Markov Models: A Brief Tutorial.* TR-98-141. ICSI, Berkeley. 1998.
3. Kanungo, Tapas. *Hidden Markov Models.* HMM tutorial slides. www.cfar.umd.edu/ kanungo.
4. Rabiner, L. and B.-H. Juang. *Fundamentals of Speech Recognition.* New Jersey: Prentice Hall. 1993.
5. Rabiner, L. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. of the IEEE* 77. 1989.
6. Duda, R. O., P. E. Hart, and D. G. Stork. *Pattern Classification.* Second Edition. Wiley-Interscience. 2001.

6 Decision Trees

Learning Objectives

After reading this chapter, you will understand

- How decision trees are used to choose the course of action
- How decision trees are used for classification
- The strengths and weakness of decision trees
- The splitting criterion used at the nodes
- What is meant by induction of decision trees
- Why pruning of decision trees is sometimes necessary

The decision tree is one of the most popularly used data structures in pattern classification. This is because of its highly transparent and user-friendly characteristics.

6.1 Introduction

A *decision tree* is a tree where each non-leaf or internal node is associated with a decision and the leaf nodes are generally associated with an outcome or class label. Each internal node tests one or more attribute values leading to two or more links or branches. Each link in turn is associated with a possible value of the decision. These links are mutually distinct and collectively exhaustive. This means that it is possible to follow only one of the links and all possibilities will be taken care of—there is a link for each possibility.

Decision trees are excellent tools for choosing between several courses of action. They provide a highly effective structure within which you can lay out options and investigate the possible outcome of choosing these options.

In the case of a binary decision tree, each node gives the statement of the decision to be taken or the comparison to be made. There are two outgoing edges from the nodes. One edge represents the outcome ''yes'' or ''true'' and the other edge, the outcome ''no'' or ''false''.

EXAMPLE 1

There are four coins a, b, c, d out of which three coins are of equal weight and one coin is heavier. Find the heavier coin.

Figure 6.1 gives the decision tree corresponding to this problem. The root node compares the weight of $a+b$ with $c+d$. If $a+b$ is heavier than $c+d$, then the outcome is ''yes'' and it takes the left branch. Otherwise, $c+d$ is heavier and it takes the right branch. The node in the left branch compares the weight of a with b. If the outcome of this comparison is ''yes'', then it chooses a as the heavier coin. If the outcome is ''no'', then it chooses b as the heavier coin. This same procedure is done with c and d if the outcome is ''no'' for the root node.

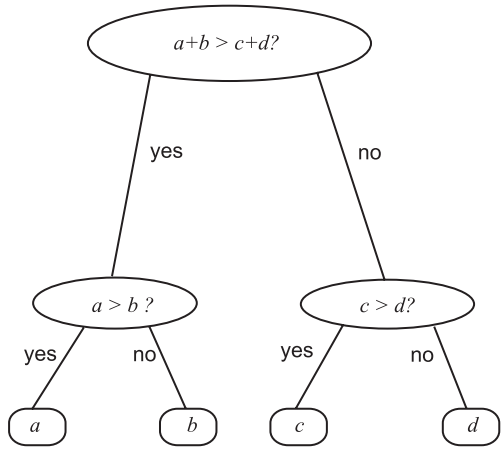


Figure 6.1 Decision tree

The example depicted in Figure 6.1 represents a simple decision tree in the context of decision making. Some of the points illustrated are:

1. There are *four leaf nodes* corresponding to the *four possible outcomes*. Each leaf node corresponds to one of the coins being heavier.
2. It requires two weighings to make the final decision or to reach a leaf node. Note that each decision node (non-terminal or internal node) corresponds to a weighing operation. Further, there are two decision nodes from the root to any leaf.
3. Each path from the root to a leaf corresponds to a rule. For example, the rule corresponding to the left-most path is ''if $a+b > c+d$ and if $a > b$ then a is heavier''.

In pattern classification, an internal node in the decision tree is associated with a

test based on the values of one or more features. These tests may be categorised as follows:

1. *Axis-parallel test* In this case, the test is of the form $x > a_0$, where x is a feature and a_0 is a threshold value. For example, ''height'' > 5 ft tests whether the height of an object is above 5 feet or not. In other words, this test is associated with a hyper-plane parallel to all the feature axes other than that of ''height'' which splits the pattern space into two parts—one part corresponds to patterns having ''height'' greater than 5 feet and the other part has patterns of ''height'' less than or equal to 5 feet. Essentially, this test involves only one feature. Such a split is called the *axis-parallel split*.
2. *Test based on linear combination of features* Here, the test is of the form

$$\sum_{i=1}^{d} a_i x_i > a_0$$

where x_i is the ith feature and a_i is the weight associated with it. This test involves a linear combination of feature values and the corresponding hyper-plane need not be parallel to any axis. For example, ''0.4 height + 0.3 weight > 38'' tests whether a linear combination of ''height'' and ''weight'' of an object exceeds 38. This test splits the space into two parts based on an *oblique* split. Note that the axis-parallel test is a special case of the oblique split where a_i is 0 for all but one i.
3. *Test based on a non-linear combination of features* This is the most general form of the test possible and it is of the form

$$f(x) > 0$$

where $f(x)$ is any non-linear function of the components of x. It is easy to see that axis-parallel and oblique splits are special cases of this test. However, this form of the test is computationally the most expensive.

It is possible to employ different types of tests at different nodes of a decision tree. However, it is computationally prohibitive to explore this possibility on large data sets. We consider only the axis-parallel and oblique tests in this chapter as they are relatively simpler to implement.

6.2 Decision Trees for Pattern Classification

Patterns can be classified using decision trees where the nodes in the tree represent the status of the problem after making some decision. The leaf nodes give the class

label of the classification rule corresponding to the path from the root node to that leaf node.

EXAMPLE 2

Let us consider a set of animals described in Table 6.1. Figure 6.2 gives a decision tree for the classification of the set of animals.

Table 6.1 Description of a set of animals

	Legs	Horns	Size	Colour	Beak	Sound
Cow	4	yes	medium	white	no	moo
Crow	2	no	small	black	yes	caw
Elephant	4	no	large	black	no	trumpet
Goat	4	yes	small	brown	no	bleat
Mouse	4	no	small	black	no	squeak
Parrot	2	no	small	green	yes	squawk
Sparrow	2	no	small	brown	yes	chirp
Tiger	4	no	medium	orange	no	roar

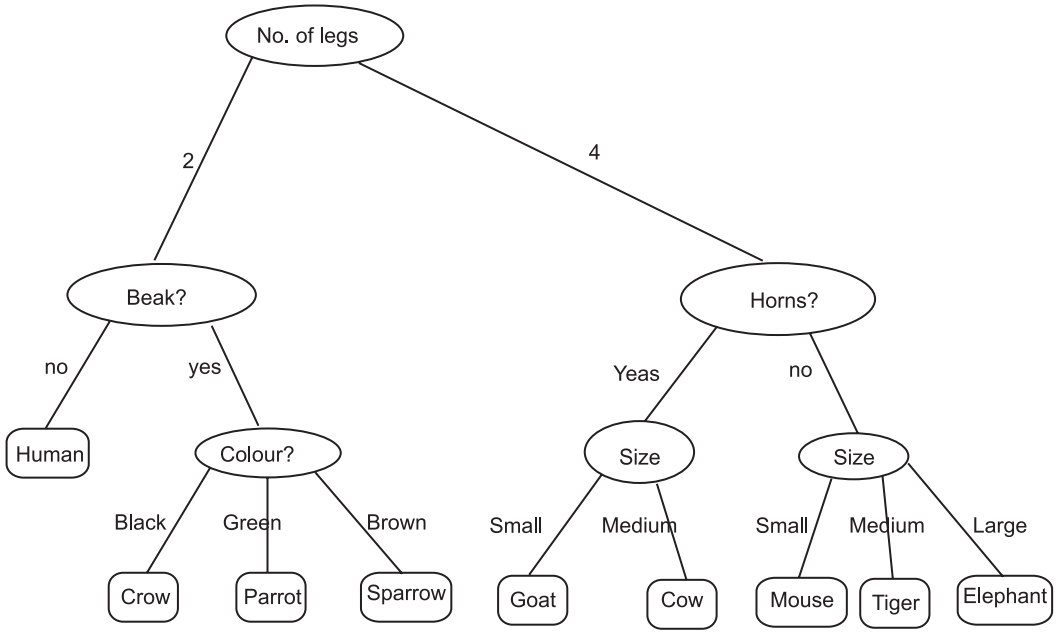


Figure 6.2 Decision tree for classification of animals

Looking at the decision tree, we can make the following observations:

- *Class labels are associated with leaf nodes* In general, leaf nodes are associated with class labels. In Figure 6.2, the leaf nodes are associated with animal names.
- *Root to leaf represents a rule* For example, looking at the decision tree in Figure 6.2, a classification rule would be:

  if (no. of legs = 4) and (has horns = false) and (size = small) then (mouse), where ''mouse'' is the class label.
- *Classification involves making a decision at every node* Classification involves making a decision at each node and moving down the appropriate branch till we reach a leaf node where the class label is determined.
- *Irrelevant features* Features that are not relevant to the classification problem at hand do not occur in the decision tree. In Table 6.1, the sound made by the animals is an attribute but does not occur in the decision tree, since it is not needed for the discrimination of the patterns.
- *Numerical and categorical features* Both numerical and categorical features can be used. We can see from the example that the colour is a categorical feature, whereas the number of legs is a numerical feature.
- *The tree can be binary or non-binary* We could have two decisions say ''yes'' and ''no'' at each node or we can have a k-ary tree, where a ''many-way decision'' may be made at a node. The colour feature has a number of branches depending on the colours considered. The horns feature has just two decisions—''yes'' or ''no''. A point to be noted here is that a many-way decision can be converted into a number of yes/no decisions and therefore *a binary tree is sufficient to represent a decision tree*.
- *A set of patterns is associated with each node* This set of patterns is larger at the top nodes and keeps reducing in size at subsequent levels. In Figure 6.2, once the decision is made on the number of legs, the node reached when ''four legs'' is the decision will contain only patterns which exclude birds and human beings.
- *The rules are simple and easy to understand* It consists of a conjunction of a number of antecedents and a single outcome. Each antecedent consists of a simple boolean function.

The animal classification problem is not a typical case of pattern classification since each pattern in the database refers to a different class. A more typical example is given below.

EXAMPLE 3

Figure 6.3 pertains to the output of a manager of a company. There are three

managers, Ram, Shyam and Mohan. Table 6.2 gives the employee records of these three managers.

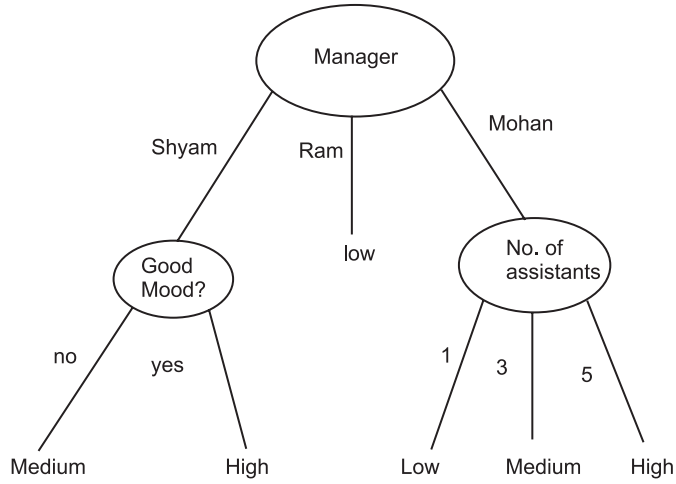


Figure 6.3 Decision tree for pattern classification

Table 6.2 Employee records in a company

Name	Age	Educational qualification	Position
Ram	55	B Com	Manager
Shyam	30	BE	Manager
Mohan	40	MSc	Manager

The decision tree in Figure 6.3 may have been induced from the patterns in Table 6.3.

Table 6.3 Patterns from which the decision tree in Figure 6.3 is induced

Manager	Number of assistants	Mood	Output
Shyam	3	No	Medium
Shyam	5	No	Medium
Shyam	1	Yes	High
Ram	1	Yes	Low
Ram	5	No	Low
Ram	5	Yes	Low
Mohan	1	No	Low
Mohan	3	Yes	Medium
Mohan	5	No	High

The output of these managers is the class labels. There are three classes: low (output), medium (output) and high (output). ''Manager'' is a categorical attribute since it pertains to the name of the manager. The ''number of assistants'' is a numerical attribute while ''mood'' is a boolean attribute which takes the values ''yes'' and ''no''. The points to be noted here are

1. *Class labels are associated with leaf nodes* The leaf nodes give one of the classes: low, medium or high.
2. *Root to leaf represents a rule* For example, looking at the decision tree in Figure 6.3, a classification rule would be

   if (Manager = Mohan) and (No. of assistants = 3) then (Output = medium)
3. *Every node involves making a decision* Every node involves making a decision which lead to two or more branches.
4. *Irrelevant features* Features that are not relevant to the classification problem at hand do not occur in the decision tree. For example, if each pattern had a feature which is the age of the manager, this feature would not appear in the decision tree—they are eliminated.
5. *Numerical and categorical features* Both numerical and categorical features appear in the decision tree.
6. *Classification is simple and efficient* It involves as many comparisons as the number of nodes, from root to leaf, in the branch followed.

Weaknesses of Decision Trees

- *Design time could be large* For example, the number of oblique splits possible could be exponential in the number of training patterns. At a single node of the tree, the complexity of selecting an optimal oblique hyper-place is exponential. If there are n training instances, each having d attributes, there are at most $2^d \times \begin{pmatrix} n \\ d \end{pmatrix}$ distinct d-dimensional oblique splits. In other words, hyper-planes can dichotomise a set of n d-dimensional vectors in at most $2 \times \sum_{k=0}^{d} \begin{pmatrix} n-1 \\ d \end{pmatrix}$ ways if $n > d+1$ and 2^n ways if $n \leq d+1$. This problem is exponential in the number of training patterns.
- *Simple (axis-parallel) decision trees do not represent non-rectangular regions well* Most decision tree algorithms only examine a single attribute at a time. This leads to rectangular classification that may not correspond well with the actual distribution of the data. In the simple decision tree, all the decision boundaries

are hyper-planes orthogonal to the axis of the feature being considered. When the regions are non-rectangular, the decision tree approximates the regions by hyper-rectangles.

EXAMPLE 4

Consider the example given in Figure 6.4. Here the decision boundary is non-rectangular. Figure 6.5 shows the approximate division performed by a simple decision tree.

- *Exponentially large decision trees are needed for some functions like the parity function and majority function*

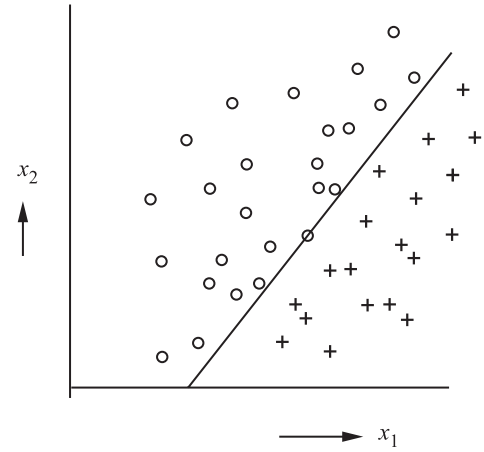


Figure 6.4 Non-rectangular decision boundary

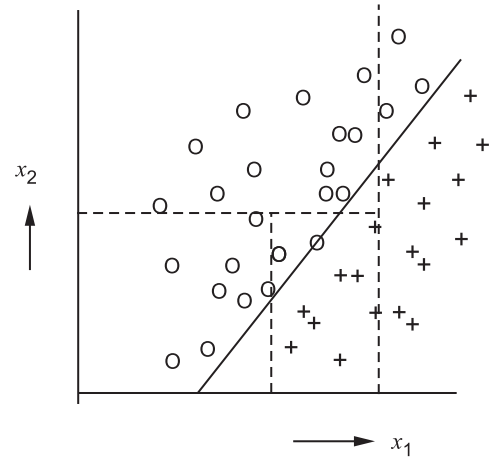


Figure 6.5 Decision tree performed on a non-rectangular region

EXAMPLE 5

A parity function takes input $x \in \{0,1\}^d$ and outputs 1 if the number of 1s in x is odd, 0 otherwise. It is evident that it is necessary to use all the d bits to decide the output. This makes the decision tree very large.

6.3 Construction of Decision Trees

A decision tree is induced from examples. We have a set of labelled patterns which forms the training set. The tree is constructed so that it agrees with the training set. The decision tree corresponding to a training set helps us to describe a large number of cases in a concise way. It is an example of inductive learning. To construct the decision tree, one or more attributes have to be chosen at each node to make a decision. The most important attribute is used at the earliest level in the decision tree. This attribute is that which makes the most difference to the classification; it is the most discriminative attribute. At each node, the set of examples is split up and each outcome is a new decision tree learning problem by itself. A really good attribute divides the examples into sets that are all positive or all negative. An attribute that leaves the example set with roughly the same proportion of positive and negative examples as the original is not good. In other words, if many of the outcomes result in a definitive answer, it is a good attribute to choose.

EXAMPLE 6

In Figure 6.6, the decision $f_1 \geq a$ divides both class 1 and class 2 so that all the patterns of each class is on one side of the boundary. The decision $f_2 \geq b$ divides the patterns of both Class 1 and Class 2 so that there are two patterns on one side and two patterns on the other side. So it can be seen that the decision $f_1 \geq a$ is a better option as it directly classifies the patterns as belonging to Class 1 or Class 2.

Once we have chosen the correct attribute, the different outcomes represent new decision trees and, in each case, the most important attribute is chosen. This is done till all nodes give a final classification.

At each node, the query which makes data to the subsequent nodes as $pure$ as possible is chosen. Thus, instead of measuring how pure the node is, we minimise the impurity of the resulting nodes. There are many measures of impurity which we shall discuss below.

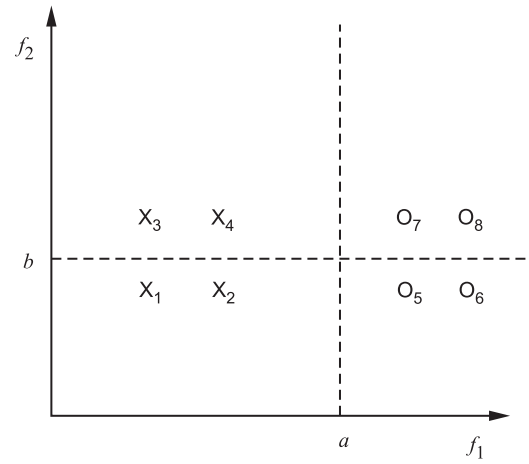


Figure 6.6 Classification into two classes: An illustration

6.3.1 Measures of Impurity

1. *Entropy impurity or information impurity*

   The entropy impurity at a node N is $i(N)$ and is given by

   $$i(N) = -\sum_j P(w_j) \log_2 P(w_j)$$

   Here $P(w_j)$ is the fraction of patterns at node N of category w_j.

   EXAMPLE 7

   Consider the case when the patterns are split equally into two sub-sets. Then $P(w_j)$ is 0.5 and we get

   $$i(N) = -0.5 \log 0.5 - 0.5 \log 0.5 = 1$$

   When all the patterns go along one branch and there are no patterns in the other branch, $i(N) = 0$. Consider 10 examples which split into three classes. The first class gets 4 examples, the second class gets 5 and the third class gets 1 example.

   $$P(w_1) = \tfrac{4}{10} = 0.4$$

   $$P(w_2) = \tfrac{5}{10} = 0.5$$

   $$P(w_3) = \tfrac{1}{10} = 0.1$$

   and $i(N) = -0.4 \log 0.4 - 0.5 \log 0.5 - 0.1 \log 0.1 = 1.36$

2. *Variance impurity*

   $i(N) = P(w_1)P(w_2)$ in a two-category case.

When $P(w_1)$ is 0.5 and $P(w_2)$ is 0.5, $i(N) = 0.25$. When $P(w_1)$ is 1.0 and $P(w_2)$ is 0, $i(N) = 0$

Generalising this to more categories, we get the *Gini impurity.*

$$i(N) = \sum_{i \neq j} P(w_i)P(w_j) = \frac{1}{2}[1 - \sum_j P^2 i(w_j)]$$

EXAMPLE 8

Again, let us we consider 10 examples which split into three classes. The first class gets 4 examples, the second class gets 5 and the third class gets 1 example. We get

$$P(w_1) = \tfrac{4}{10} = 0.4$$

$$P(w_2) = \tfrac{5}{10} = 0.5$$

$$P(w_3) = \tfrac{1}{10} = 0.1$$

We can then calculate the Gini impurity as

$$i(N) = \tfrac{1}{2}[1 - 0.4^2 - 0.5^2 - 0.1^2] = 0.29$$

3. *Misclassification impurity*

$$i(N) = 1 - \max_j P(w_j)$$

EXAMPLE 9

In the example we have considered with three branches, the misclassification impurity will be

$$i(N) = 1 \max(0.4, 0.5, 0.1) = 0.5$$

This measures the minimum probability that a training pattern would be misclassified.

6.3.2 Which Attribute to Choose?

The attribute to be chosen should decrease the impurity as much as possible. The drop in impurity is defined as

$$\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R)$$

This is in the case when there are only two branches at the node. Here P_L and N_L

correspond to the left branch and N_R corresponds to the right branch. If there are more than two branches, we get

$$\Delta i(N) = i(N) - \sum_j P_j \times i(N_j)$$

j takes on the value for each of the outcomes of the decision made at the node. $\Delta i(N)$ can also be called the gain in information at the node.

The attribute which maximises $\Delta i(N)$ is to be chosen.

EXAMPLE 10

Consider a case where there are 100 examples with 40 belonging to C_1, 30 belonging to C_2 and 30 belonging to C_3. Let some attribute X split these examples into two branches, the left branch containing 60 examples and the right branch containing 40 examples. The left branch contains 40 examples of C_1, 10 examples of C_2 and 10 examples of C_3. The right branch contains 0 examples of C_1, 20 examples of C_2 and 20 examples of C_3. This is shown in Table 6.4.

Table 6.4 Number of examples in the decision tree after a split using feature X

X = a left branch	X = b right branch	Total	Class
40	0	40	1
10	20	30	2
10	20	30	3

Using entropy impurity

The entropy impurity at the split is

$$\begin{aligned} i(N) &= -\frac{40}{100}\log\frac{40}{100} - \frac{30}{100}\log\frac{30}{100} - \frac{30}{100}\log\frac{30}{100} \\ &= -0.4\log 0.4 - 0.3\log 0.3 - 0.3\log 0.3 = 1.38 \end{aligned}$$

The entropy of the left branch $i(N_L)$ is

$$i(N_L) = -\frac{40}{60}\log\frac{40}{60} - \frac{10}{60}\log\frac{10}{60} - \frac{10}{60}\log\frac{10}{60} = 1.25$$

The entropy of the right branch $i(N_R)$ is

$$i(N_R) = -\frac{20}{40}\log\frac{20}{40} - \frac{20}{40}\log\frac{20}{40} = 1.0$$

The drop in impurity is therefore

$$\Delta i(N) = 1.38 - (\frac{60}{100} \times 1.25) - (\frac{40}{100} \times 1.0)$$
$$= 1.38 - 0.75 - 0.4 = 0.23$$

Using variance impurity

The impurity at the node is

$$i(N) = \frac{1}{2}(1 - 0.4^2 - 0.3^2 - 0.3^2)$$
$$= 0.5 \times 0.66 = 0.33$$

The impurity at the left node is

$$i(N_L) = \frac{1}{2}(1 - 0.6667^2 - 0.1667^2 - 0.1667^2)$$
$$= 0.5 \times 0.5 = 0.25$$

The impurity at the right node is

$$i(N_R) = \frac{1}{2}(1 - 0.5^2 - 0.5^2)$$
$$= 0.5 \times 0.5 = 0.25$$

The drop in impurity is therefore

$$\Delta i(N) = 0.33 - (\frac{60}{100} \times 0.25) - (\frac{40}{100} \times 0.25) = 0.08$$

Using misclassification impurity

The impurity at the node is

$$i(N) = 1 \max(0.4, 0.3, 0.3) = 0.6$$

Impurity at the left node is

$$i(N_L) = 1 \max(\frac{40}{60}, \frac{10}{60}, \frac{10}{60})$$
$$= 1 \max(0.6667, 0.1667, 0.1667) = 0.333$$

Impurity at the right node is

$$i(N_L) = 1 \max(\frac{20}{40}, \frac{20}{40})$$
$$= 1 \max(0.5, 0.5) = 0.5$$

The drop in impurity is therefore

$$\Delta i(N) = 0.6 - (\frac{60}{100} \times 0.33) - (\frac{40}{100} \times 0.5)$$
$$= 0.6 - 0.198 - 0.2 = 0.202$$

6.4 Splitting at the Nodes

Each decision outcome at a node is called a split since the training data is split into sub-sets. The root node splits the full training data. Each successive decision splits the data at the node into proper sub-sets.

The decision rule at each non-leaf node is of the form

$$f_i(x) > a_0$$

This is called the split rule or test at the node. Depending on this split rule, we can have different types of splitting.

1. *Axis-parallel split* It pertains to splitting depending on the value of a single attribute and is of the form

   $$x_j > a_0$$

   Here the decision is taken depending on the attribute x_j. If it is greater than a_0, one branch is taken, otherwise the other branch is taken.

   EXAMPLE 11

   Axis parallel split is illustrated in Figure 6.7. Since each decision involves only one feature, the division of the coordinate space is parallel to each axis.

   $X_1 = (1, 1)$; $X_2 = (2, 1)$
   $X_3 = (1, 2)$; $X_4 = (2, 2)$
   $O_5 = (6, 1)$; $O_6 = (7, 1)$
   $O_7 = (6, 2)$; $O_8 = (7, 2)$
   $X_9 = (6, 7)$; $X_{10} = (7, 7)$

   The rule at the first node in the axis-parallel split would be $f_1 > 4$ and the rule at the node at the next level would be $f_2 > 4$ resulting in the decision tree shown in Figure 6.8.

Figure 6.7 shows 10 points whose coordinates are as follows:

2. *Oblique split* In the case of the oblique split (Figure 6.9), with the general form of the equations being $a.f_1 + b.f_2 + c > 0$, we get the following equations:

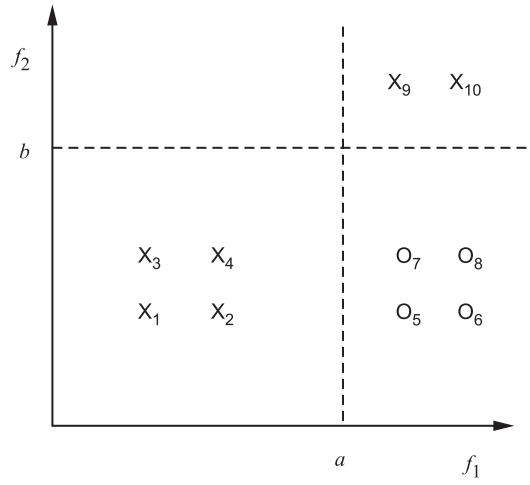


Figure 6.7 Axis-parallel split

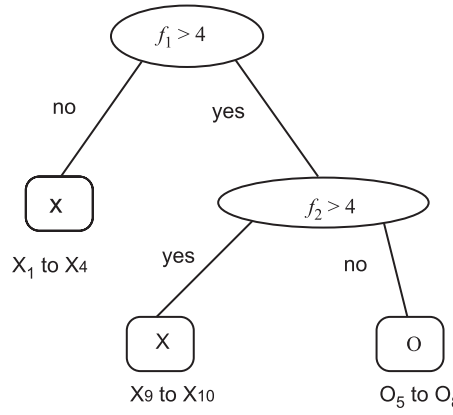


Figure 6.8 Final decision tree for the patterns shown in Figure 6.7 using axis-parallel split

$$2a + b + c > 0$$

$$7a + 7b + c > 0$$

$$6a + 2b + c < 0$$

If we solve these inequalities, we get $a = -1$, $b = 1$ and $c = 2$.

The rule in oblique split therefore becomes $-f_1 + f_2 + 2 > 0$ or $f_1 - f_2 < 2$, giving the decision tree shown in Figure 6.10.

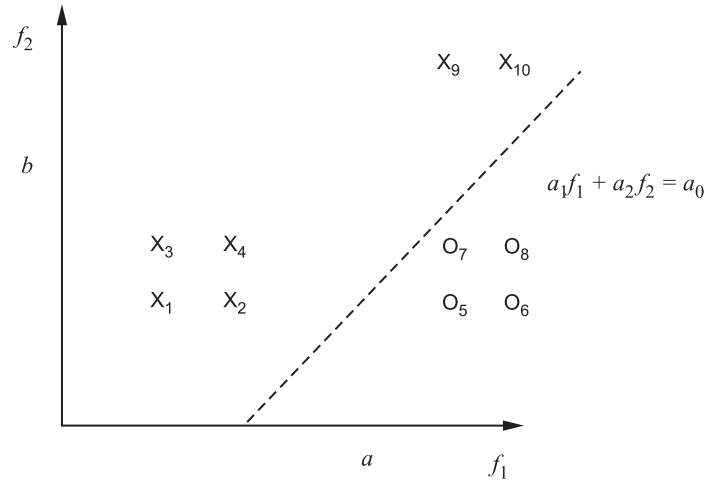


Figure 6.9 A two-class example to illustrate oblique split

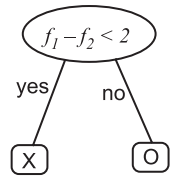


Figure 6.10 Decision tree for the patterns shown in Figure 6.9 using oblique split

3. *Multivariate split* Multivariate means the use of more than one attribute for the split at a node as opposed to univariate splitting. While a linear combination of features gives us the oblique split, it is also possible to have a non-linear combination of features to affect the split at a node.

6.4.1 When to Stop Splitting

1. *Use cross-validation*
   A part of the training data is kept aside for validation. This can also be done repeatedly keeping different sub-sets for validation and taking the average performance. The decision tree which gives the best results during validation is the tree to be considered.

2. *Reduction in impurity*
   Splitting can be stopped, when the reduction in impurity is very small. This means

   $$\max_s i(s) \leq \beta,$$

   where β is a small value.

3. *Depending on the global criterion function*
   If the criterion function is of the form

   $$\alpha \times \text{size} + \sum_{\text{leafnodes}} i(N)$$

   where α is a positive constant and size = number of nodes or links, the strategy is to stop splitting when this global criterion reaches some minimum value.

6.5 Overfitting and Pruning

Whenever there is a large set of possible hypotheses, the tree can keep growing—thus becoming too specific. An example of this is having one unique path through the tree for every pattern in the training set. This is called overfitting. The solution to this problem of overfitting is to use the technique of decision tree pruning. Pruning prevents recursive splitting using attributes that are not clearly relevant.

6.5.1 Pruning by Finding Irrelevant Attributes

In this method, the decision tree is first constructed fully and is then pruned by finding irrelevant attributes. When we split a set of examples using an attribute, the resulting sub-sets may be roughly the same proportion of each class as the original set. This results in zero information gain which is a clue to the irrelevance of the attribute. Looking at the data and the number of positive and negative examples, we can calculate the extent to which it deviates from a perfect absence of pattern. If the degree of deviation is statistically unlikely, then this is considered good evidence for the presence of a significant pattern in the data. The deviation in terms of number of positive and negative examples p_i and n_i and the expected numbers $\tilde{p}_i$ and $\tilde{n}_i$ will be

$$D = \sum_{i=1}^{v} \frac{(p_i - \tilde{p}_i)^2}{\tilde{p}_i} + \frac{(n_i - \tilde{n}_i)^2}{\tilde{n}_i}$$

where v is the sample size. D is distributed according to χ^2 (chi-squared) distribution. The probability that the attribute is really irrelevant can be calculated with the help of the standard χ^2 table. This is called χ^2 pruning.

6.5.2 Use of Cross-Validation

Overfitting can be prevented by cross-validation explained in Section 6.4.1. This is the process of estimating how well the current hypothesis will predict unseen data. Using cross-validation, it is possible to select a tree with a good prediction performance.

6.6 Example of Decision Tree Induction

A simple problem on pattern classification is considered and the decision tree is induced.

EXAMPLE 12

Table 6.5 Example training data set for induction of a decision tree

Cook	Mood	Cuisine	Tasty
Sita	Bad	Indian	Yes
Sita	Good	Continental	Yes
Asha	Bad	Indian	No
Asha	Good	Indian	Yes
Usha	Bad	Indian	Yes
Usha	Bad	Continental	No
Asha	Bad	Continental	No
Asha	Good	Continental	Yes
Usha	Good	Indian	Yes
Usha	Good	Continental	No
Sita	Good	Indian	Yes
Sita	Bad	Continental	Yes

Table 6.5 gives a set of training data. In this data, there are two classes, Tasty = yes and Tasty = no. Eight examples correspond to Tasty = yes and 4 examples to Tasty = no. The information of the data set is therefore

$$i(N) = -\frac{4}{12}\log\frac{4}{12} - \frac{8}{12}\log\frac{8}{12} = 1.8089$$

Let us consider the three attributes and find the attribute with the highest gain.

1. *Cook*

   (a) Cook = Sita has four examples belonging to Tasty = yes and 0 examples belonging to Tasty = no. This branch has an entropy of 0.

(b) Cook = Asha has 2 examples belonging to tasty = yes and 2 examples belonging to tasty = no. The entropy for Cook = Asha is

$$i(N_A) = -\tfrac{2}{4}\log\tfrac{2}{4} - \tfrac{2}{4}\log\tfrac{2}{4} = 1.0$$

(c) Cook = Usha has 2 examples belonging to Tasty = yes and 2 examples belonging to Tasty = no. The entropy for Cook = Usha is therefore

$$i(N_U) = -\frac{2}{4}\log\frac{2}{4} - \frac{2}{4}\log\frac{2}{4} = 1.0$$

The gain is therefore

$$\text{Gain(Cook)} = 1.8089 - \frac{4}{12} \times 1.0 - \frac{4}{12} \times 1.0 = 1.4223$$

2. *Mood*

(a) Mood = bad has three examples belonging to Tasty = yes and 3 examples belonging to Tasty = no. The entropy for Mood = bad is therefore

$$i(N_B) = -\frac{3}{6}\log\frac{3}{6} - \frac{3}{6}\log\frac{3}{6} = 1.0$$

(b) Mood = good has 5 examples belonging to Tasty = yes and 1 example belonging to Tasty = no. The entropy for Mood = good is therefore

$$i(N_G) = -\frac{1}{6}\log\frac{1}{6} - \frac{5}{6}\log\frac{5}{6} = 2.4588$$

The gain is therefore

$$\text{Gain(Mood)} = 1.8089 - \frac{6}{12} \times 2.4588 - \frac{6}{12} \times 1.0 = 0.0795$$

3. *Cuisine*

(a) Cuisine = Indian has 5 examples belonging to Tasty = yes and 1 example belonging to Tasty = no. The entropy for Cuisine = Indian is therefore

$$i(N_I) = -\frac{1}{6}\log\frac{1}{6} - \frac{5}{6}\log\frac{5}{6} = 2.4588$$

(b) Cuisine = Continental has 3 examples belonging to Tasty = yes and 3 examples belonging to Tasty = no. The entropy for Cuisine = Continental is therefore

$$i(N_C) = -\frac{3}{6}\log\frac{3}{6} - \frac{3}{6}\log\frac{3}{6} = 1.0$$

The gain is therefore

$$\text{Gain(Cuisine)} = 1.8079 - \frac{6}{12} \times 2.4588 - \frac{6}{12} \times 1.0 = 0.0795$$

The attribute with the largest gain is Cook and therefore Cook is chosen as the first attribute in the decision tree.

1. *Cook = Sita*

   Cook = Sita has 4 examples belonging to Tasty = yes and 0 examples for Tasty = no, giving an entropy of 0. This branch has reached the leaf node.

2. *Cook = Asha*

   Cook = Asha has 2 examples belonging to Tasty = yes and 2 examples for Tasty = no. The entropy for Cook = Asha is therefore

$$I(N) = -\frac{2}{4}\log\frac{2}{4} - \frac{2}{4}\log\frac{2}{4} = 1.0$$

   (a) Mood

   Mood = bad has 2 examples belonging to tasty = no and 0 examples belonging to Tasty = yes, giving an entropy of 0. Mood = good has 2 examples belonging to Tasty = yes and 0 examples belonging to Tasty = no, giving an entropy of 0. The gain for Mood will therefore be 1.0.

   (b) Cuisine

   Cuisine = Indian has 1 examples belonging to Tasty = yes and 1 examples belonging to Tasty = no. The entropy for Cuisine = Indian will therefore be

$$I(N) = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} = 1.0$$

   Cuisine = Continental has 1 example belonging to Tasty = yes and 1 example belonging to Tasty = no. The entropy for Cuisine = Continental will therefore be

$$I(N) = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} = 1.0$$

   The gain for Cuisine is

$$\text{Gain(Cuisine)} = 1.0 - \frac{2}{4} \times 1.0 - \frac{2}{4} \times 1.0 = 0$$

Since Mood has a higher gain, it is chosen as the next attribute after Cook = Asha.

3. *Cook = Usha*

   If a similar process is followed for Cook = Usha, the attribute with the highest gain is Cuisine and therefore Cuisine is the next attribute chosen after Cook = Usha. (Refer to Exercise Problem No. 8.)

The final decision tree is as shown in Figure 6.11. It can be seen from the figure that if Cook = Sita, we do not require information about mood or cuisine to get the right classification. If Cook = Asha, information about the type of cuisine is redundant and for Cook = Usha, information about the mood is redundant. The leaf nodes represent the class labels.

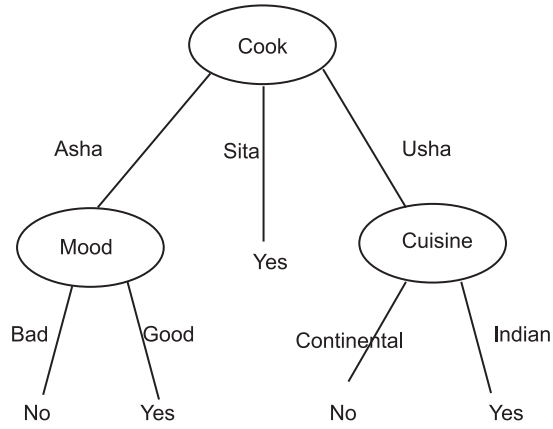


Figure 6.11 Decision tree induced from training examples

Discussion

The decision tree is an effective tool for choosing between several courses of action. It corresponds to simple rules which represent each path in the decision tree. It is induced from examples by finding which attribute gives the highest drop in impurity or highest increase in gain at each decision point.

Further Reading

Quinlan (1986) explains the basic principle of the decision tree with an example. He further describes a software for decision trees called C4.5 (1992; 1996). The

various types of splitting rules for decision trees is discussed by Buntine and Niblett (1992). Murthy et al. (1994), in their paper, describe the obligue split in decision trees. The OC1 decision tree software system is also developed and described in this paper. Using a search technique, John (1994) finds multivariate splits in decision trees. Sethi and Yoo (1994) talk of a multivariate split decision tree. Wang et al. (2008) discuss fuzzy decision trees. Important papers have been written by Chandra and Varghese (2009), Kalkanis (1993), Yildiz and Dikmen (2007), Chen, Wu and Tang (2009), Ouyang et al. (2009), Chen, Hu and Tang (2009), Sieling (2008) and Twala et al. (2008)

Exercises

1. Construct a decision tree when one of three coins a, b, c is counterfeit—two are of equal weight and the third is either heavier or lighter.
2. Give an example of a decision tree where the following types of splitting criteria are used.
   (a) Axis-parallel split
   (b) Oblique split
   (c) Multivariate split
3. When a decision tree is induced from the training set, some features may not be used. Explain and give an example.
4. Induce a decision tree for the problem of choosing which restaurant to visit. The features to be used are
   (a) Expense: Expensive, reasonable, cheap
   (b) Location from home: Far, near, very close
   (c) Quality of food: Very good, fair, bad
   (d) Weather: Raining, sunny, overcast
5. Consider an axis-parallel decision tree built using n training patterns. Assume that there are C classes. What is the maximum depth of a binary decision tree corresponding to this data? What is the minimum depth of the binary decision tree?
6. Consider the following two-dimensional patterns:

Class 1	Class 2
(1, 1)	(6, 1)
(1, 2)	(6, 2)
(2, 1)	(1, 8)
	(2, 7)
	(2, 8)

(a) Obtain the best axis-parallel tree using entropy gain.

(b) Is an oblique tree simpler than the axis-parallel tree?

7. Consider a decision tree where there are a total of 25 examples. In the first node, these examples are split so that 10 examples pertain to the decision variable of the left branch and 15 to the right branch. Further, the left branch is split into three branches so that of 10, 0 pertains to the first branch, 4 examples to the second branch and 6 examples to the third branch. Find the gain of each of the branches.

8. Consider the decision tree induced for the examples given in Table 6.3. Complete the decision tree for Cook = Usha.

Computer Exercises

1. Write a program to design a decision tree and induce a decision tree for the example in Table 6.4.

2. Explore the different impurity measures to calculate gain. Implement them in the decision tree construction program and compare them.

3. Implement axis-parallel split and oblique split. Incorporate them in the decision tree construction program.

Bibliography

1. Buntine, W. and T. Niblett. A further comparison of splitting rules for decision-tree Induction. *Machine Learning* 8: 75–85. 1992.

2. Chandra, B. and P. Paul Varghese. Moving towards efficient decision tree construction. *Information Sciences* 179(8): 1059–1069. 2009.

3. Chen, Yen-Liang, Chia-Chi Wu and Kwei Tang. Building a cost-constrained decision tree with multiple condition attributes. *Information Sciences* 179(7): 967–079. 2009.

4. Chen, Yen-Liang, Hsiao-Wei Hu and Kwei Tang. Constructing a decision tree from data with hierarchical class labels. *Expert Systems with Applications* 36(3) Part 1: 4838–4847. 2009.

5. John, George H. Finding multivariate splits in decision trees using function optimization. *Proceedings of the AAAI*. 1994.

6. Kalkanis, G. The application of confidence interval error analysis to the design of decision tree classifiers. *Pattern Recognition Letters* 14(5): 355–361. 1993.

7. Murthy, Sreerama K., Simon Kasif and Steven Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research* 2: 1–32. 1994.

8. Ouyang, Jie, Nilesh Patel and Ishwar Sethi. Induction of multiclass multifeature split decision trees from distributed data. *Pattern Recognition*. 2009.

9. Quinlan, J. R. Induction of decision trees. *Machine Learning* 1: 81–106. 1986.

10. Quinlan, J.R. *C4.5-Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann. 1992.

11. Quinlan, J.R. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research* 4: 77–90. 1996.

12. Sethi, Ishwar K. and Jae H. Yoo. Design of multicategory multifeature split decision trees using perceptron learning. *Pattern Recognition* 27(7): 939–947. 1994.

13. Sieling, Detlef. Minimization of decision trees is hard to approximate. *Journal of Computer and System Sciences* 74(3): 394–403. 2008.

14. Twala, B. E. T. H., M. C. Jones and D. J. Hand. Good methods for coping with missing data in decision trees. *Pattern Recognition Letters* 29(7): 950–956. 2008.

15. Wang, Xi-Zhao, Jun-Hai Zhai and Shu-Xia Lu. Induction of multiple fuzzy decision trees based on rough set technique. *Information Sciences* 178(16): 3188–3202. 2008.

16. Yildiz, Olcay Taner and Onur Dikmen. Parallel univariate decision trees. *Pattern Recognition Letters* 28(7): 825–832. 2007.

7 Support Vector Machines

Learning Objectives

After reading this chapter, you will

- Learn how to classify patterns using a linear decision boundary
- Understand the intricacies of the perceptron learning algorithm and how it is used to learn weight vectors
- Discover neural networks and how they were evolved from studying the human brain
- Know what is meant by support vector machines

7.1 Introduction

A support vector machine (SVM) is a binary classifier. It abstracts a decision boundary in multi-dimensional space using an appropriate sub-set of the training set of vectors; the elements of this sub-set are the support vectors. Geometrically, support vectors are those training patterns that are closest to the decision boundary. In order to appreciate the behaviour of SVMs, it is useful to understand several associated concepts including linear discriminant functions and neural networks. So, we introduce these concepts before describing SVMs.

7.1.1 Linear Discriminant Functions

Linear discriminant functions can be used to discriminate between patterns belonging to two or more classes. We illustrate this using the collection of two-dimensional patterns shown in Figure 7.1.

EXAMPLE 1

There are *five* patterns from one class (labelled X) and *four* patterns from the second class (labelled O). This set of labelled patterns may be described using Table 7.1.

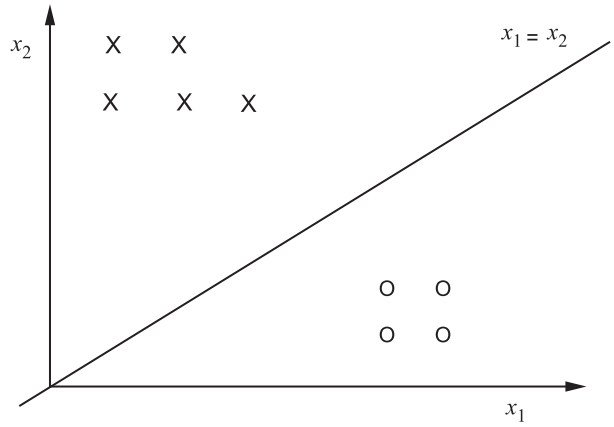


Figure 7.1 Classification using a linear discriminant function

Table 7.1 Description of the patterns

Pattern no.	1	2	Class
1	0.5	3.0	X
2	1	3	X
3	0.5	2.5	X
4	1	2.5	X
5	1.5	2.5	X
6	4.5	1	O
7	5	1	O
8	4.5	0.5	O
9	5.5	0.5	O

The first five patterns are from class X and the remaining patterns are from class O. Consider the line represented by $x_1 = x_2$ shown in Figure 7.1. All the patterns labelled X are to the left of the line and patterns labelled O are on the right side.

In other words, patterns corresponding to these two classes are *linearly separable* as patterns belonging to each of the classes fall on only one side of the line or equivalently, a line can separate the patterns belonging to class X from those of class O.

One more way of separating the patterns of the two classes is by noting that all the patterns of class X satisfy the property that $x_1 < x_2$ or equivalently $x_1 - x_2$ is less than zero.

For example, pattern 1 in Table 7.1 has a value of 0.5 for x_1 and 3 for x_2 and so this property is satisfied (because $0.5 - 3 = -2.5(< 0)$). Similarly, the fifth pattern satisfies this property because $x_1 - x_2$ is -1. In a symmetric manner, all the patterns belonging to class O have the value of x_1 to be larger than that of x_2; equivalently, $x_1 - x_2 > 0$. For example, for the ninth pattern in the table, $x_1 - x_2$ is 5 ($5.5 - 0.5$) which is greater than 0.

Based on the location of the patterns with respect to the line, we say that patterns of class O are on the positive side of the line (because $x_1 - x_2 > 0$ for these patterns) and patterns of class X are on the negative side (equivalently $x_1 - x_2 < 0$ for patterns labelled X). We consider the remaining patterns, that is patterns numbered 2 to 4 and 6 to 8 in Exercise 1 at the end of the chapter. Here, the line separates the two-dimensional space into two parts which can be called *positive half space* (where patterns from class O are located) and *negative half space* (which has patterns from class X).

It is easy to observe that there are possibly infinite ways of realising the decision boundary, a line in this two-dimensional case, which can separate patterns belonging to the two classes. For example, the line $x_1 - x_2 = 1$ also separates the first five points in Table 7.1 from the remaining 4 points. Here, $x_1 - x_2$ is less than 1 for the patterns labelled X and greater than 1 for the patterns labelled O.

Note that the two lines $x_1 - x_2 = 0$ and $x_1 - x_2 = 1$ are parallel to each other. Obviously, the line $x_1 - x_2 = 0$ passes through the origin, whereas $x_1 - x_2 = 1$ shifts away from the origin to provide more space for class X. Another line that separates the patterns labelled X from those labelled O is $3x_1 - 2x_2 = 0$; this is not parallel to the earlier lines. It is convenient to abstract all such lines using the following functional form:

$$f(x) \quad = w_1 x_1 + w_2 x_2 + b = 0 \tag{7.1}$$

Correspondingly, the line $x_1 - x_2 = 0$ has $w_1 = 1$; $w_2 = -1$; and $b = 0$. Similarly, $x_1 - x_2 = 1$ has $w_1 = 1$; $w_2 = -1$; and $b = -1$. This representation permits us to deal with patterns and linear decision boundaries in multi-dimensional space in a flexible manner. For example, in a d-dimensional space, the decision boundary is a hyper-plane and can be represented by

$$f(x) \quad = w^t x + b = 0 \tag{7.2}$$

where w and x are d-dimensional vectors. When $b = 0$, we call it a homogeneous form of representation; otherwise, it is a non-homogeneous form. We can use this representation to characterise linear separability. We say that two classes, say classes labelled ''X'' and ''O'' are linearly separable, if we can find a weight vector w and a scalar b such that

$$w^t x + b > 0$$

for all patterns x belonging to one class (say class ''O'') and

$$w^t x + b < 0$$

for all the patterns belonging to the other class (that is class ''X''). We will illustrate this using the data shown in Figure 7.1.

EXAMPLE 2

Consider the 9 two-dimensional patterns shown in Figure 7.1. Let us further consider the linear classifier based on the vector $w^t = (1, -1)$ and $b = -1$. We provide in Table 7.2, details of the patterns, their labels, and the value of $w^t x + b$ for each pattern. Observe that the values (of $w^t x + b$) are negative (< 0) for the first five patterns which are labelled ''X'' and are positive (> 0) for all the four patterns labelled ''O'' showing that the two classes are linearly separable.

Table 7.2 Linearly separable classes

Pattern No.	x_1	x_2	Class	$w^t x + b$
1	0.5	3.0	X	−3.5
2	1	3	X	−3.0
3	0.5	2.5	X	−3.0
4	1	2.5	X	−2.5
5	1.5	2.5	X	−2.0
6	4.5	1	O	2.5
7	5	1	O	3.0
8	4.5	0.5	O	3.0
9	5.5	0.5	O	4.0

Here, the value of b plays a role in deciding the location of the decision boundary. Note that the decision boundary between the two classes is characterised by $w^t x + b = 0$. Based on the location of the origin (the zero vector), we can make the following observations with respect to the value of b.

1. Consider the situation where $b = 0$. In such a case, the origin lies on the decision boundary. This is because $w^t x + b = 0$ for $x = 0$ (origin) and $b = 0$, for any value of w.

2. Consider the value of b to be positive, that is $b > 0$. Here, the origin lies in the positive side; this is because $w^t x + b > 0$ when $x = 0$ and $b > 0$, for any value of w.

3. When the value of b is negative, that is, when $b < 0$, the origin lies in the negative side; this can be observed from the fact that $w^t x + b < 0$ when $x = 0$ and $b < 0$, for any value of w.

The above observations hold in general in d-dimensional ($d >= 1$) space. We will examine them with respect to the two-dimensional data set ($d = 2$) shown in Example 1.

EXAMPLE 3

Consider the data shown in Figure 7.1 and Table 7.1. Let us consider the line $x_1 = x_2$; here, $w^t = (1, -1)$ and $b = 0$. The origin (0, 0) is on the line $x_1 = x_2$ which is the decision boundary.

Now consider the line $2x_1 - 2x_2 = -1$; this can also be written as $2x_1 - 2x_2 + 1 = 0$. This line is obtained by shifting $x_1 - x_2 = 0$ up appropriately (refer to Exercise 2). Note that the points $(-0.5, 0)$ and $(0, 0.5)$ are on this line. Also, all the points labelled ''X'' are on one side (negative side) and those labelled ''O'' are located on the other side (positive side) of the line. Note that the origin is in the positive side.

The line $2x_1 - 2x_2 = 1$ or equivalently, $2x_1 - 2x_2 - 1 = 0$ is obtained by shifting down the line $x_1 = x_2$. Here, the origin is in the negative half portion. Points (0.5, 0) and (1, 0.5) are located on $2x_1 - 2x_2 = 1$. Also, it classifies all the nine labelled patterns shown in Table 7.1 correctly.

So, the value of b in some sense characterises the location of the decision boundary. It characterises the extent of shift of the decision boundary (in this case a line) from the origin. Similarly, the weight vector w decides the orientation of the decision boundary. This may be explained using an example. Even though we have considered a two-dimensional example to illustrate the concepts in a simple manner, these conclusions naturally apply to multi-dimensional situations. In such a case, the decision boundary is a hyper-plane.

EXAMPLE 4

Consider the two-dimensional data shown in Figure 7.1. Let us consider the line $x_1 = x_2$ (or equivalently $x_1 - x_2 = 0$) as the decision boundary. The corresponding $w^t = (1, -1)$; $w_1 = 1$ and $w_2 = -1$. Note that w is orthogonal to the decision boundary. Consider any point x on the decision boundary; it is of the form (α, α). So, $w^t x = 0$. Further, consider shifting the decision boundary up or down to get $2x_1 - 2x_2 = -1$ or $2x_1 - 2x_2 = 1$ respectively. Note that while considering a line parallel to $x_1 = x_2$ as the decision boundary, the weight vector does not change; only the value of b changes appropriately. So, we can infer that the vector w is orthogonal to the decision boundary. It is possible to extend a similar argument to deal with patterns in a d-dimensional space (for $d > 2$ also; refer to Exercise 3).

So, w is orthogonal to the decision boundary; it is directed towards the positive half space. This means that any pattern x in the positive side (patterns of class O) will make an angle θ with w such that $-90 < \theta < 90$ which is equivalent to having $\cos\theta > 0$. This may be illustrated using the following example.

EXAMPLE 5

Consider the two-dimensional data shown in Figure 7.1. Let us consider the line

$x_1 = x_2$ as the decision boundary. The corresponding $w^t = (1, -1)$. Consider any pattern belonging to the class ''O''; let us say, we consider pattern 7 in Table 7.1. The cosine of the angle between w and a pattern x is $\frac{w^t x}{||w \,||||x||}$. The value of $w^t x$ is 4 (note that it is $5 - 1$) and is positive. The cosine of the angle between w and x, when x is the pattern numbered 7, is $\frac{4}{\sqrt{2}\,\sqrt{26}}$ which is positive. This means that w is orthogonal to the decision boundary and points to the positive half space.

Another useful notion is the distance between a point x and the decision boundary. Any point x can be written as a sum of two vectors; one vector along the decision boundary and another orthogonal to it. So,

$$x = x_b + x_o \tag{7.3}$$

where x_b is the projection of x along the decision boundary and x_o is the orthogonal component. Further, we know that w is also orthogonal to the decision boundary and is oriented towards the positive half space. So, x_o is of the form $p\frac{w}{|w|}$, where p is a real number and is positive if x is from class O and negative if it is from class X. Based on these observations, we have the following:

$$f(x) = w^t x + b = w^t x_b + b + w^t x_o = 0 + w^t x_o = p w^t \frac{w}{|\, w \,|} = p|| \, w \, || \tag{7.4}$$

From the above equation, we have

$$p = \frac{w^t x + b}{|| \, w \, ||} \tag{7.5}$$

We call p the normal or the shortest distance between x and the decision boundary. We will now simply denote p as the distance and provide an example to illustrate computation of (normal or shortest) distance between a vector and the decision boundary.

EXAMPLE 6

Consider the data shown in Figure 7.1 and the decision boundary given by the line $x_1 - x_2 - 1 = 0$. Note that in this case, $|| \, w \, ||^2$ is 2 and $b = -1$. So, distance from the origin ($x = (0, 0)$) is $\frac{-1}{\sqrt{2}}$.

Similarly, distance from the point (1,1) is also $\frac{-1}{\sqrt{2}}$. In general, it can be shown that (refer to Exercise 5) the distance from any point of the form (α, α), where α is a real number is $\frac{-1}{\sqrt{2}}$.

Also, we can transform the patterns from a d-dimensional space to a $d + 1$-dimensional space by adding 1 as the $d + 1$th component to each pattern, that is, x_{d+1} = 1, and by considering $w_{d+1} = b$. This mapping permits us to represent the decision boundary in the *homogeneous* form as

$$f(x') \quad = z^t x' = 0 \tag{7.6}$$

where z and x' are $(d+1)$-dimensional vectors; so, z is obtained by adding b as the $d+1$th value to w and x' is obtained by adding to x, a value of 1 as the $d+1$th entry. So, z and x' are given as follows:

$$z = \begin{pmatrix} w \\ b \end{pmatrix}$$

and

$$x' = \begin{pmatrix} x \\ 1 \end{pmatrix}$$

Using this representation, the line $x_1 - x_2 = 1$, that is, $x_1 - x_2 - 1 = 0$, can be represented as $z^t x' = 0$, where $z^t = (1, -1, -1)$ and $x'^t = (x_1, x_2, 1)$.

In the rest of the chapter, we will use w and x instead of z and x' respectively. The distinction between the homogeneous case and the non-homogeneous case would be obvious from the context.

In the binary classification problem, it is possible to affect normalisation so that all the patterns lie on the positive side of the line. This is achieved by replacing every feature value of each pattern from class X by its negation including the $d+1$th entry.

EXAMPLE 7

The first pattern in Table 7.1, $(0.5, 3)^t$ after transformation is $(0.5, 3, 1)^t$ and after normalisation is $(-0.5, -3, -1)$. So, by transforming and normalising the data in Table 7.1 in this manner, we get the data shown in Table 7.3.

Table 7.3 Description of normalised 3-dimensional patterns

Pattern	1	2	3
x_1	−0.5	−3.0	−1
x_2	−1	−3	−1
x_3	−0.5	−2.5	−1
x_4	−1	−2.5	−1
x_5	−1.5	−2.5	−1
x_6	4.5	1	1
x_7	5	1	1
x_8	4.5	0.5	1
x_9	5.5	0.5	1

It is convenient to use the normalised data to automatically learn the decision boundary when the classes are linearly separable. We will introduce the *perceptron* learning algorithm for learning the linear discriminant function in the next section.

7.2 Learning the Linear Discriminant Function

In the previous section, we have seen that there can be different decision boundaries (lines) that separate the patterns belonging to two different classes. In general, it is possible to obtain the decision boundary in the form of a hyper-plane if we learn the vector w. Let the dimensionality of the pattern space be d. So, in the non-homogeneous case ($b \neq 0$), w is a $(d+1)$-dimensional vector with b as the $d+1$th value.

Correspondingly, for the homogeneous case, the $d+1$th component of w is 0. In either case, the $d+1$th value of x is chosen to be 1. This would permit us to work with the same representation either for the homogeneous case or the non-homogeneous case.

It is simpler to learn the weight vector w when the classes are linearly separable. We describe an algorithm, called the *perceptron learning algorithm*, for learning the weight vector when the classes are linearly separable.

7.2.1 Learning the Weight Vector

This algorithm considers patterns sequentially and updates the weight vector w if a pattern is misclassified by the weight vector. It iterates through the set of patterns till there is no updation required, or equivalently, no pattern is misclassified during an entire iteration. It is convenient to consider patterns in the normalised form while applying this algorithm. This would mean that a pattern x is misclassified if $w^t x \leq 0$. In order to use the homogeneous form, we need to consider the patterns after transformation and normalisation. The set of 3-dimensional vectors corresponding to the patterns given in Table 7.1 is given in Table 7.3. The patterns are labelled as x_1 to x_9. The problem is to find a 3-dimensional weight vector w which classifies all the patterns correctly. Equivalently, w should be such that $w^t x$ is greater than 0 for all the nine patterns in Table 7.1.

In the perceptron learning algorithm, for learning w, we let the normalised patterns in the $(d+1)$-dimensional space to be $x_1, x_2, \cdots, x_n$. Note that the patterns in the d-dimensional space have been mapped to the $(d+1)$-dimensional space as discussed in the previous section.

Perceptron Learning Algorithm

STEP 1: Initialise w as the null vector. In other words, set $i = 0$ and $w_i = 0$.

STEP 2: For $j = 1$ to n do
if $w_i^t x_j \leq 0$ then $i = i + 1$ and $w_i = w_{i-1} + x_j$.

STEP 3: Repeat Step 2 till the value of i does not change during an entire iteration over all the n patterns.

It is possible to explain the intuition behind the proposed scheme for updating the weight vector as follows. Let w_i be the current weight vector which has misclassified the pattern x_j; that is, $w_i^t x_j < 0$. So, w_{i+1} is obtained as $w_i + x_j$ using the above algorithm. Note that $w_{i+1}^t x_j$ is equal to $w_i^t x_j$ + $|| x_j ||^2$. As a consequence, $w_{i+1}^t x_j$ is larger than $w_i^t x_j$ by $|| x_j ||^2$ because $|| x_j ||^2$ is positive. This would mean that the updated vector w_{i+1} is better suited than w_i to classify x_j correctly because $w_{i+1} x_j$ is larger than $w_i x_j$ and so can be positive even if $w_i x_j$ were not.

EXAMPLE 8

We can illustrate the working of the algorithm using the data in Table 7.3 with the help of the following steps.

1. $w_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$ and $x_1 = \begin{pmatrix} -0.5 \\ -3.0 \\ 1 \end{pmatrix}$

   Here, $w_1^t x_1 = 0$ and so $w_2 = w_1 + x_1$ which is represented by

   $$w_2 = \begin{pmatrix} -0.5 \\ -3 \\ -1 \end{pmatrix}$$

2. Next we consider pattern x_2. $w_2^t x_2$ is 10.5 (> 0). Similarly, x_3, x_4, and x_5 are also properly classified.

   Note that $w_2^t x_3 = 8.75$, $w_2^t x_4 = 9$, and $w_2^t x_5 = 9.25$. So, these patterns are properly classified by w_2 and do not affect the weight vector.

3. However, $w_2^t x_6 = -6.25$ (< 0). So, $w_3 = w_2 + x_6$, that is

   $$w_3 = \begin{pmatrix} 4 \\ -2 \\ 0 \end{pmatrix}$$

   Note that w_3 classifies patterns x_7, x_8, x_9, and in the next iteration, x_1, x_2, x_3, and x_4 correctly. Specifically, $w_3^t x_7 = 18$, $w_3^t x_8 = 17$, $w_3^t x_9 = 21$, $w_3^t x_1 = 4$, $w_3^t x_2 = 2$, $w_3^t x_3 = 3$, $w_3^t x_4 = 1$.

4. However x_5 is misclassified by w_3. Note that $w_3^t x_5$ is -1 (< 0). So, $w_4 = w_3 + x_5$ is obtained. It is given by

   $$w_4 = \begin{pmatrix} 2.5 \\ -4.5 \\ -1 \end{pmatrix}$$

w_4 classifies patterns x_6, x_7, x_8, x_9, x_1, x_2, x_3, x_4, and x_5 correctly. Specifically, $w_4^t x_6 = 5.75$, $w_4^t x_7 = 7$, $w_4^t x_8 = 8$, $w_4^t x_9 = 10.5$, $w_4^t x_1 = 13.25$, $w_4^t x_2 = 11.5$, $w_4^t x_3 = 11$, $w_4^t x_4 = 9.75$, and $w_4^t x_5 = 8.5$.

5. So w_4 is the desired vector. In other words, $2.5x_1 - 4.5x_2 - 1 = 0$ is the equation of the decision boundary; equivalently, the line separating the two classes is $5x_1 - 9x_2 - 2 = 0$.

In general, it is possible to show that the perceptron learning algorithm will converge to the correct weight vector in a finite number of iterations when the classes are linearly separable. The number of iterations may increase based on the location of the training patterns. This will be illustrated in the next sub-section.

7.2.2 Multi-class Problems

A linear discriminant function is ideally suited to separate patterns belonging to two classes that are linearly separable. However, in real life applications, there could be a need to classify patterns from *three* or more classes. It is possible to extend a binary classifier, in different ways, to classify patterns belonging to C classes, where $C > 2$. We list some of the popularly used schemes below:

1. Consider a pair of classes at a time. Note that there are $\frac{C(C-1)}{2}$ such pairs when there are C classes in the given collection. Learn a linear discriminant function for each pair of classes. Combine these decisions to make the overall decision. We illustrate this scheme using a two-dimensional example.

EXAMPLE 9

Consider the labelled data shown in Table 7.4.

Table 7.4 Description of the patterns of 3 classes

Pattern No.	1	2	Class
1	0.5	3	X
2	1	3	X
3	0.5	2.5	X
4	6	6	O
5	6	6.5	O
6	7	6	O
7	10	0.5	*
8	10	1	*
9	11	1	*

Here, we have 9 labelled patterns with 3 patterns from each of the classes labelled ‘‘X’’, ‘‘O’’, and ‘‘*’’. Further, each pattern is a point in a

two-dimensional space. If we consider all the possible binary classifiers (dealing with two classes at a time), we have three possibilities. One possibility is a classifier dealing with classes ''X'' and ''O''. The second one deals with the classes ''X'' and ''*'' and finally the third one deals with ''O'' and ''*''. We will consider these three binary classifiers in turn below.

(a) *Separation between "X" and "O"* For this, we consider the 6 data points in Table 7.4 corresponding to these two classes. For classifying into either ''X'' (positive class) or ''O (negative class), the transformed and normalised data is shown in Table 7.5.

Table 7.5 Data for X vs O

Pattern No.	1	2	Bias
1	0.5	3	1
2	1	3	1
3	0.5	2.5	1
4	−6	−6	−1
5	−6	−6.5	−1
6	−7	−6	−1

Using the perceptron learning algorithm, we start with

i.
$$w_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$
and
$$x_1 = \begin{pmatrix} 0.5 \\ 3 \\ 1 \end{pmatrix}$$
Here, $w_1^t x_1 = 0$ and so $w_2 = w_1 + x_1$ which is represented by
$$w_2 = \begin{pmatrix} 0.5 \\ 3 \\ 1 \end{pmatrix}$$

ii. Next we consider pattern x_2 and $w_2^t x_2$ is 10.5 (> 0). Similarly, x_3 is also properly classified as $w_2^t x_3$ is 8.75 (> 0).

iii. Note that $w_2^t x_4$ is -22 (< 0). So, we get $w_3 = w_2 + x_4$ and it is
$$w_3 = \begin{pmatrix} -5.5 \\ -3 \\ 0 \end{pmatrix}$$

Note that w_3 classifies patterns x_5 and x_6 correctly.

iv. However, w_3 fails to classify x_1 correctly as $w_3^t x_1$ is -11.75 (< 0). So, we update w_3 to get $w_4 = w_3 + x_1$ and it is given by

$$w_4 = \begin{pmatrix} -5 \\ 0 \\ 1 \end{pmatrix}$$

v. Now we consider x_2. $w_4^t x_2$ is -4 (< 0). So, we update w_4 and get $w_5 = w_4 + x_2$ and so

$$w_5 = \begin{pmatrix} -4 \\ 3 \\ 2 \end{pmatrix}$$

Observe that w_5 correctly classifies patterns x_3, x_4, x_5, x_6, x_1, and x_2. So, w_5 is the desired vector. This weight vector characterises the decision boundary (a line in this case) separating the two classes ''X'' and ''O''. So, we call it w_{xo}.

(b) *Separation between "X" and "*"* For this, we consider the 6 data points given in Table 7.4 corresponding to these two classes. For classifying into either ''X'' (positive class) and ''*'' (negative class), the transformed and normalised data is shown in Table 7.6.

Table 7.6 Data for X vs *

Pattern No.	1	2	bias
1	0.5	3	1
2	1	3	1
3	0.5	2.5	1
4	−10	−0.5	−1
5	−10	−1	−1
6	−11	−1	−1

Here, by using the perceptron learning algorithm, we start with

$$w_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

get

$$w_2 = \begin{pmatrix} 0.5 \\ 3 \\ 1 \end{pmatrix}$$

then get

$$w_3 = \begin{pmatrix} -9.5 \\ 2.5 \\ 0 \end{pmatrix}$$

and finally get (refer to Exercise 9)

$$w_4 = \begin{pmatrix} -9 \\ 5 \\ 1 \end{pmatrix}$$

Note that w_4 classifies all the 6 patterns in Table 7.6 correctly. So we call it w_{x*} and

$$w_{x*} = \begin{pmatrix} -9 \\ 5 \\ 1 \end{pmatrix}.$$

(c) *Classification between "O" and "*"* For this, we consider the 6 data points in Table 7.4 corresponding to these two classes. For classifying into either ''O'' (positive class) and ''*'' (negative class), the transformed and normalised data is shown in Table 7.7.

Table 7.7 Data for O vs *

Pattern No.	1	2	Bias
1	6	6	1
2	6	6.5	1
3	7	6	1
4	−10	−0.5	−1
5	−10	−1	−1
6	−11	−1	−1

Here, by using the perceptron learning algorithm, we start with

$$w_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

and end up with (refer to Exercise 10)

$$w_3 = \begin{pmatrix} -4 \\ 5.5 \\ 0 \end{pmatrix}$$

and it classifies all the patterns in Table 7.7 correctly. So,

$$w_{o*} = \begin{pmatrix} -4 \\ 5.5 \\ 0 \end{pmatrix}$$

Observe that the 3 binary classifiers are characterised by the three vectors w_{xo}, w_{x*}, and w_{o*}. Now if we take these three vectors and consider the patterns in Table 7.4, we can classify the patterns as shown below.

(a) Let us consider pattern x_1 (after transformation) given by

$$x_1 = \begin{pmatrix} 0.5 \\ 3 \\ 1 \end{pmatrix}$$

It may be observed that $w_{xo}^t x_1 = 9\ (> 0)$. So, x_1 is classified as belonging to class ''X''. Similarly, $w_{x*}^t x_1 = 11.5\ (> 0)$ confirming that x_1 belongs to class ''X''. Further, $w_{o*}^t x_1 = 14.5\ (> 0)$ which suggests that x_1 is from class ''O''. However, out of the three binary classifiers, a majority (two out of three) of the decisions are in favour of assigning pattern x_1 to class ''X''. So, we assign x_1 to ''X''.

(b) Let us consider pattern x_5 given by

$$x_5 = \begin{pmatrix} 6 \\ 6.5 \\ 1 \end{pmatrix}$$

It may be observed that $w_{xo}^t x_5 = -2.5\ (< 0)$. So, x_5 is classified as belonging to class ''O''. Next, $w_{x*}^t x_5 = -20.5 (< 0)$ suggesting that x_5 belongs to class ''*''. Further, $w_{o*}^t x_5 = 11.75\ (> 0)$ which suggests that x_5 is from class ''O''. However, out of the three binary classifiers, a majority (two out of three) of the decisions are in favour of assigning pattern x_5 to class ''O''. So, we assign x_5 to ''O''.

(c) Now let us consider the pattern x_9 given by

$$x_9 = \begin{pmatrix} 11 \\ 1 \\ 1 \end{pmatrix}$$

It may be observed that $w_{xo}^t x_9 = -39\ (< 0)$. So, x_9 may be classified as belonging to class ''O''. Next, $w_{x*}^t x_9 = -93\ (< 0)$ suggesting that x_9 belongs to class ''*''. Further, $w_{o*}^t x_9 = -38.5\ (< 0)$ which suggests that x_9 is from class ''*''. Out of the three binary classifiers, a majority (two out of three) of the decisions are in favour of assigning pattern x_9 to class ''*''. So, we assign x_9 to ''*''.

It is possible to classify the remaining 6 patterns in the table in a similar manner (refer to Exercise 11).

2. Consider two-class problems of the following type. For each class C_i, create the class $\overline{C}_i$ which consists of patterns from all the remaining classes. So, $\overline{C}_i = \bigcup_{j=1;j\neq i}^{C} C_j$. Learn a linear discriminant function to classify each of these two-class problems. Note that there are C such two-class problems. These C linear discriminants give us the overall decision. We illustrate this scheme using the following example.

EXAMPLE 10

Consider the two-dimensional data shown in Table 7.4. There are three classes and the possible binary classifiers are considered below.

1. *Separation between "X" and "$\overline{X}$" (rest of the classes)* The transformed and normalised data set is shown in Table 7.8.

   In order to obtain the decision boundary, we use the perceptron training algorithm on the two classes, namely, "X" and "$\overline{\text{X}}$".

   (a) We start with

$$w_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Table 7.8 Classification of X from the rest

Pattern No.	1	2	Bias
1	0.5	3	1
2	1	3	1
3	0.5	2.5	1
4	−6	−6	−1
5	−6	−6.5	−1
6	−7	−6	−1
7	−10	−0.5	−1
8	−10	−1	−1
9	−11	−1	−1

and

$$x_1 = \begin{pmatrix} 0.5 \\ 3 \\ 1 \end{pmatrix}$$

Here, $w_1^t x_1 = 0$ and so $w_2 = w_1 + x_1$ which is represented by

$$w_2 = \begin{pmatrix} 0.5 \\ 3 \\ 1 \end{pmatrix}$$

Observe that w_2 correctly classifies x_2 and x_3.

(b) However, w_2 fails to classify x_4 correctly as $w_2^t x_4$ is -22 (< 0). So, w_2 gets updated to give $w_3 = w_2 + x_4$ and it is

$$w_3 = \begin{pmatrix} -5.5 \\ -3 \\ 0 \end{pmatrix}$$

w_3 classifies patterns x_5, x_6, x_7, x_8, and x_9 correctly.

(c) However, w_3 classifies x_1 incorrectly. So, it gets updated to give

$$w_4 = \begin{pmatrix} -5 \\ 0 \\ 1 \end{pmatrix}$$

(d) w_4 misclassifies x_2. So, it is updated to give

$$w_5 = \begin{pmatrix} -4 \\ 3 \\ 2 \end{pmatrix}$$

Now it is easy to observe that w_5 correctly classifies patterns x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_1, and x_2. So, w_5 is the desired vector as it correctly classifies all the patterns in Table 7.8. We denote w_5 as w_x because it separates the patterns of ''X'' from the rest.

2. *Separation of "O" from "$\overline{O}$" (the rest)* In this case, we consider patterns of class ''O'' as positively labelled and the patterns of the other two classes (''X'' and ''*'') as negatively labelled. So, by using appropriate transformation and normalisation of the patterns given in Table 7.4, we get the data shown in Table 7.9.

   Here, we use the perceptron training algorithm to separate patterns of class ''O'' from the rest.

   (a) We start with

$$w_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

and

$$x_1 = \begin{pmatrix} -0.5 \\ -3 \\ -1 \end{pmatrix}$$

Here, $w_1^t x_1 = 0$ and so $w_2 = w_1 + x_1$ which is represented by

$$w_2 = \begin{pmatrix} -0.5 \\ -3 \\ -1 \end{pmatrix}$$

Continuing with the algorithm gives us the desired weight vector w_o after several iterations. It is given by (refer to Exercise 11)

$$w_o = \begin{pmatrix} -1.5 \\ 5.5 \\ -21 \end{pmatrix}$$

Note that w_o correctly classifies all the patterns in Table 7.9.

Table 7.9 Classification of O from the rest

Pattern No.	1	2	Bias
1	−0.5	−3	−1
2	−1	−3	−1
3	−0.5	−2.5	−1
4	6	6	1
5	6	6.5	1
6	7	6	1
7	−10	−0.5	−1
8	−10	−1	−1
9	−11	−1	−1

(b) *Separation of class "*" from "$\overline{*}$" (the rest)* Here, we use the perceptron training algorithm to obtain a weight vector w_* that separates patterns of class ''*'' from the rest. Here, class ''*'' is positive and the rest are negative. So, the transformed and normalised data for this classification is shown in Table 7.10.

Here, we start with

$$w_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Table 7.10 Classification of * from the rest

Pattern No.	1	2	Bias
1	−0.5	−3	−1
2	−1	−3	−1
3	−0.5	−2.5	−1
4	−6	−6	−1
5	−6	−6.5	−1
6	−7	−6	−1
7	10	0.5	1
8	10	1	1
9	11	1	1

and after four updates get the weight vector (refer to Exercise 14)

$$w_5 = \begin{pmatrix} 2.5 \\ -11.5 \\ -2 \end{pmatrix}$$

Further, w_5 classifies all the patterns in Table 7.10 correctly. So,

$$w_* = \begin{pmatrix} 2.5 \\ -11.5 \\ -2 \end{pmatrix}$$

Observe that the 3 binary classifiers are characterised by the three vectors w_x, w_o, and w_*. Now if we take these three vectors and consider the patterns in Table 7.4, we can classify the patterns as shown below.

1. Let us consider pattern x_1 (after transformation) given by

$$x_1 = \begin{pmatrix} 0.5 \\ 3 \\ 1 \end{pmatrix}$$

It may be observed that $w_x^t x_1 = 9\ (> 0)$. So, x_1 is classified as belonging to class ''X''. Similarly, $w_0^t x_1 = -5.25\ (< 0)$ confirming that x_1 does not belong to class ''O''. Further, $w_*^t x_1 = -35.25\ (< 0)$ which suggests that x_1 is not from class ''*''. So, we assign x_1 to ''X''.

2. Let us consider pattern x_5 given by

$$x_5 = \begin{pmatrix} 6 \\ 6.5 \\ 1 \end{pmatrix}$$

It may be observed that $w_x^t x_5 = -2.5\ (< 0)$. So, x_5 is not a member of class ''X''. Next, $w_o^t x_5 = 5.75\ (> 0)$ suggesting that x_5 belongs to class ''O''.

Further, $w_*^t x_5 = -71.75$ (< 0) which suggests that x_5 is not from class ''*''. So, x_5 is assigned to ''O''.

3. Now let us consider the pattern x_9 given by

$$x_9 = \begin{pmatrix} 11 \\ 1 \\ 1 \end{pmatrix}$$

It may be observed that $w_x^t x_9 = -39$ (< 0). So, x_9 does not belong to class ''X''. Next, $w_o^t x_9 = -32$ (< 0) suggesting that x_9 does not belong to class ''O'' either. Further, $w_*^t x_9 = 14$ (> 0) which suggests that x_9 is from class ''*''. So, we assign x_9 to ''*''.

It is possible to classify the remaining 6 patterns in the table in a similar manner (refer to Exercise 15).

It is also possible to characterise Boolean functions using a perceptron. We discuss training a perceptron for a Boolean OR (inclusive or represented by $\vee$) function in the next example.

EXAMPLE 11

Consider the truth table of Boolean OR given in Table 7.11.

Table 7.11 Truth table of Boolean OR

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

There are two classes ''0'' and ''1'' corresponding to the output values 0 and 1 of $x_1 \vee x_2$ respectively. There is one pattern of class ''0'' and three patterns of class ''1''. After transformation and normalisation, we have the four patterns as shown in Table 7.12 (refer to Exercise 16).

Table 7.12 Transformed and normalised data for Boolean OR

0	0	-1
0	1	1
1	0	1
1	1	1

In order to train the perceptron, we go through the following steps.

1. $w_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$

   and

   $$x_1 = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

   Here, $w_1^t x_1 = 0$ and so $w_2 = w_1 + x_1$ which is represented by

   $$w_2 = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}$$

2. $w_2^t x_2 = -1 < 0$. So, update w; $w_3 = w_2 + x_2$ which is given by

   $$w_3 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

3. Going through the sequence of updates, we get

   $$w_{10} = \begin{pmatrix} 2 \\ 2 \\ -1 \end{pmatrix}$$

   Note that w_{10} classifies all the four patterns correctly (refer to Exercise 17)

7.2.3 Generality of Linear Discriminants

The notion of linear discriminant functions is very general. This idea may be extended to handle even non-linear discriminants using homogeneous representation. For example,

$$f(x) \quad = 1 + x + x^2 = 0 \tag{7.7}$$

may be represented as

$$f(x') \quad = z^t x' = 0 \tag{7.8}$$

where

$$z = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$x' = \begin{pmatrix} x^2 \\ x \\ 1 \end{pmatrix}$$

It may be illustrated using the following example.

EXAMPLE 12

Consider a binary classifier which assigns pattern x to class ''O'' (positive class) if $f(x) > 0$ and to class ''X'' (negative class) if $f(x) < 0$, where

$$f(x) \quad = a + bx + cx^2 \tag{7.9}$$

Note that based on the discussion above, we can equivalently assign x to ''O'' if $z^t x' > 0$ and to class ''X'' if $z^t x' < 0$, where

$$z = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

$$x' = \begin{pmatrix} 1 \\ x \\ x^2 \end{pmatrix}$$

Let us consider a set of labelled patterns that are not linearly separable. Specifically, let us consider the one-dimensional data set shown in Table 7.13.

Table 7.13 Non-linearly separable data

Pattern No.	x	Class
1	1	O
2	−1	O
3	2	O
4	−2	O
5	3	X
6	4	X
7	−3	X
8	−4	X

Observe that the data is not linearly separable. Further, the decision boundary is characterised by $f(x) = 0$. Now by appropriate transformation and normalisation, where the components in x' are 1, value of x, and value of x^2, we get the data shown in Table 7.14.

Table 7.14 Normalised non-linearly separable data

Pattern No.	1	x	x^2	Bias
1	1	1	1	1
2	1	−1	1	1
3	1	2	4	1
4	1	−2	4	1
5	−1	3	−9	−1
6	−1	4	−16	−1
7	−1	−3	−9	−1
8	−1	−4	−16	−1

We can use the perceptron learning algorithm to obtain the weight vector z.

1. Here, we start with

$$z_1 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

and

$$x'_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

z_1 misclassifies x'_1. So, it gets updated to $z_2 = z_1 + x'_1$.

2. Continuing with the algorithm, we end up with z_{27} which classifies all the 8 patterns in Table 7.14 correctly. It is given by

$$z_{27} = \begin{pmatrix} 12 \\ -1 \\ -4 \\ 12 \end{pmatrix}$$

So, the decision boundary is given by

$$f(x) \quad = 12 - x - 4x^2 + 12 = 0 \tag{7.10}$$

This example illustrates the generality of linear discriminants. It is possible to deal with non-linearly separable classes using a linear discriminant. Further, it is also possible to extend this idea to vector-valued x. Neural networks are popular tools for learning such generalised linear discriminants. We deal with neural networks in the next section.

7.3 Neural Networks

An artificial neural network was first evolved by looking at how the human brain works. The human brain has millions of neurons which communicate with other neurons using electrochemical signals. Signals are received by neurons through junctions called synapses. The inputs to a neuron are combined in some way and if it is above a threshold, the neuron fires and an output is sent out to other neurons through the axon. This principle is also used in artificial neural networks. From now on, we use ''neural networks'' to address artificial neural networks as well.

The output of the neural network depends on the input and the weights in the network. The training of the neural network consists of making the network give the correct output for every input. It starts by taking random weights for every link in the network. When an input is given to the network, the output is observed. If the output is correct, then nothing is done to the weights in the network. If the output is wrong, the error is calculated and it is used to update all the weights in the network. This procedure is carried out for a large number of inputs, till the output is the correct output for every input. Learning in neural networks is nothing but making appropriate changes in weights.

7.3.1 Artificial Neuron

The neural network here consists of artificial neurons. These neurons are modelled in the same way as the neurons in the human brain. The input to the neuron is weighted and summed up and if this aggregate exceeds a threshold, the neuron outputs a signal. For example, if it is above the threshold, it could output a 1; otherwise it could output a 0. An artificial neuron is shown diagrammatically in Figure 7.2.

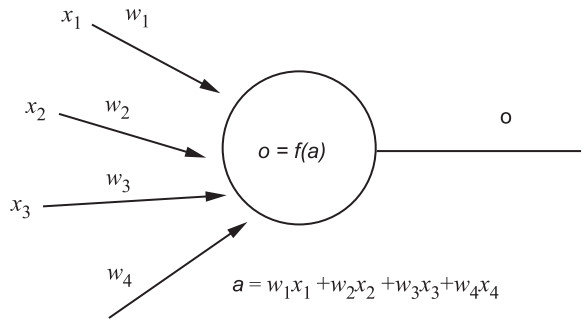


Figure 7.2 Patterns represented in feature space

If $x_1, x_2, \cdots, x_n$ are the inputs to the neuron with the corresponding weights $w_1, w_2, \cdots, w_n$, the activation will be

$$a = w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

The output, o, of the neuron is a function of this activation. One popular activation function used is the threshold function shown in Figure 7.3 where t is the threshold. This can be written as

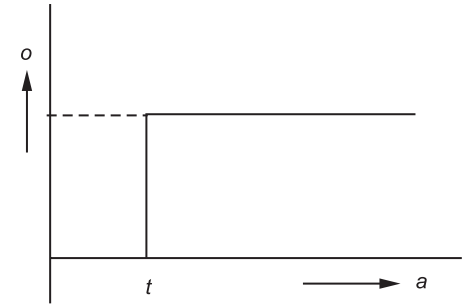


Figure 7.3 Threshold activation function

$$\text{Output } = o = 1 \text{ if } a \geq t$$
$$\text{Output } = o = 0 \text{ if } a < t$$

We have used such an activation function in perceptron learning. Consider the problem examined in Example 8. The equation of the decision boundary is $5x_1 - 9x_2 - 2 = 0$ or equivalently $5x_1 - 9x_2 = 2$. Consider the corresponding original data patterns given in Table 7.1. It is easy to verify that for each pattern labelled X,

$$5x_1 - 9x_2 < 2$$

and for each pattern of class O

$$5x_1 - 9x_2 > 2.$$

The corresponding perceptron is shown in Figure 7.4.

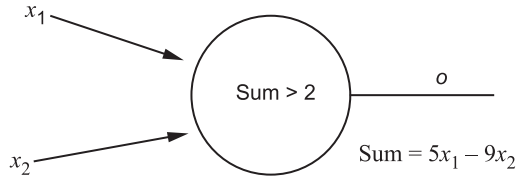


Figure 7.4 Perceptron for the problem in Example 8

7.3.2 Feed-forward Network

This is the simplest network which has input units and output units. All the nodes of the input units are connected to the output units. The output from the output units is the output of the network. This is shown in Figure 7.5.

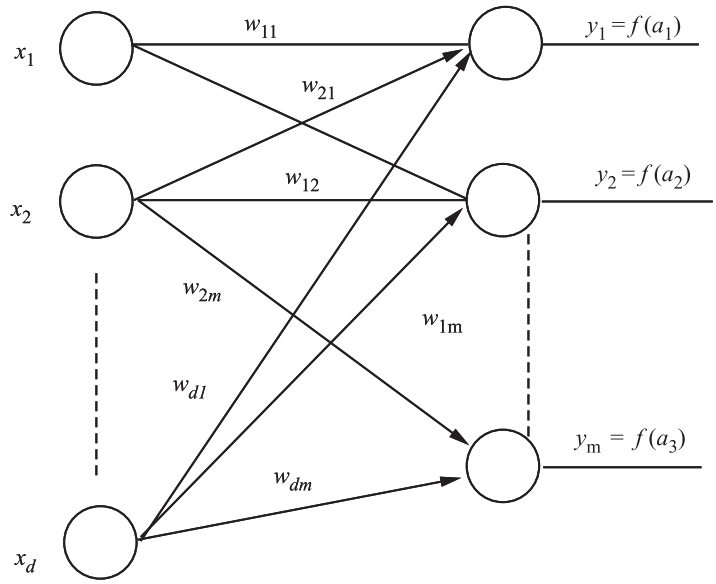


Figure 7.5 Feed-forward network

Let the input layer have n nodes $I_1, I_2, \cdots, I_d$. The output nodes are $O_1, O_2, \cdots, O_m$. The inputs are $x_1, x_2, \cdots, x_d$. The weights from the input layer to the output layer are $w_{11}, w_{12}, \cdots, w_{1m}, \cdots, w_{d1}, w_{d2}, \cdots, w_{dm}$. The outputs are $y_1, y_2, \cdots, y_m$. Note that the perceptron network shown in Figure 7.4 is a feed-forward network with $d = 2$ and $m = 1$. The corresponding weights are $w_{11} = 5$ and $w_{21} = -9$. We used the threshold activation function with the threshold $t = 2$.

Training

The weights of the network are initially set randomly using numbers between 0 and 1. When an input is given to the network, the activation at the output unit i ($1 \leq i \leq m$) will be

$$a_i = w_{1i}x_1 + w_{2i}x_2 + \cdots + w_{di}x_d$$

And the output of the ith unit will be $o_i = f(a_i)$.

When a training pattern is input to the network, its correct output (or target) will be known. Let it be $t_1, \cdots, t_m$. The error at the ith output node will be $e_i = t_i - o_i$. These error values are used for back-propagation which updates the weights of the network. The weights are updated as follows. Weight w_{ji} between the jth input node and the ith output node will be updated as follows:

$$w_{ji} = w_{ji} + \alpha \times x_j \times e_i \tag{7.11}$$

We illustrate these ideas using the following example.

EXAMPLE 13

Consider the data given in Table 7.15. There are four patterns. Each is four-dimensional ($d = 4$) and there are two target outputs ($m = 2$).

Table 7.15 Four-dimensional data with two output units

Pattern No.	x_1	x_2	x_3	x_4	t_1	t_2
1	1	0	0	0	1	0
2	0	1	0	0	1	0
3	0	0	0	1	0	1
4	0	0	1	0	0	1

1. Let us assume that all the weights are initialised to a value of 0.5. That is, $w_{ji} = 0.5$ (for $1 \leq j \leq 4$ and $i = 1, 2$).
2. Let us consider the first pattern for which $x_1 = 1$; $x_2 = x_3 = x_4 = 0$. So, $a_1 = 0.5$ and $a_2 = 0.5$. We use the threshold activation function with threshold $t = 0$. So, $o_1 = o_2 = 1$; and $e_1 = t_1 - o_1 = 0$ and $e_2 = t_2 - o_2 = -1$.
3. We can update the weights using $\alpha = 0.5$ and equation (7.11) as follows:

$$w_{11} = 0.5 + 0.5 \times 1 \times 0 = 0.5; w_{12} = 0.5 + 0.5 \times 1 \times (-1) = 0$$
$$w_{21} = 0.5 + 0.5 \times 0 \times 0 = 0.5; w_{22} = 0.5 + 0.5 \times 0 \times (-1) = 0.5$$
$$w_{31} = 0.5 + 0.5 \times 0 \times 0 = 0.5; w_{32} = 0.5 + 0.5 \times 0 \times (-1) = 0.5$$
$$w_{41} = 0.5 + 0.5 \times 0 \times 0 = 0.5; w_{42} = 0.5 + 0.5 \times 0 \times (-1) = 0.5$$

4. Now consider the second pattern where $x_2 = 1$; $x_1 = x_3 = x_4 = 0$. After updating the weights, we observe that there is no change in the values of w_{ji} but for w_{22} whose value becomes 0 (Exercise 18).
5. After presenting the remaining two patterns and updating the weights, we end up with the following weights (Exercise 18).

$$w_{11} = w_{21} = 0.5; w_{31} = w_{41} = 0$$
$$w_{12} = w_{22} = 0; w_{32} = w_{42} = 0.5$$

6. Using these weights, it can seen that all the four patterns are correctly classified. So, $e_1 = e_2 = 0$ and weights will not change for each of the four patterns (Exercise 18).

7.3.3 Multilayer Perceptron

The simple feed-forward network discussed in the previous section can deal with classification problems where the classes are linearly separable. It cannot represent non-linear decision boundaries. For example, consider the data shown in Table 7.16. It shows the truth table of the Boolean connective ''exclusive OR''. There are two inputs ($d = 2$) and one output ($m = 1$). It is a two-class problem corresponding to the truth values ''0'' and ''1'' of g. Let $w_{11} = a$ and $w_{21} = b$, where a and b are real numbers.

Table 7.16 Truth table for exclusive OR

x_1	x_2	$g(x_1, x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

Assuming a threshold value of c, we have to show that $ax_1 + bx_2 > c$ for the class ''1'' and $ax_1 + bx_2 \leq c$ for patterns in class ''0''. So, we get the following inequalities corresponding to the four patterns in the table.

$$a.0 + b.0 \leq c \tag{7.12}$$
$$a.0 + b.1 > c \tag{7.13}$$
$$a.1 + b.0 > c \tag{7.14}$$
$$a.1 + b.1 \leq c \tag{7.15}$$

From equation (7.12), we know that c is non-negative. From equations (7.14) and (7.13), we have a and b as positive and each of them greater than c. However, equation (7.15) requires that $a + b$ be less than or equal to c which clearly violates equations (7.12), (7.13), and (7.14). So, a feed-forward network cannot handle non-linear decision boundaries. In order to deal with such non-linear functions, we require multilayer networks. Besides the input and output units, they also contain either one or two layers of hidden nodes. Due to the hidden layer(s), even non-linear functions can be handled. Figure 7.6 shows a multilayer perceptron with one hidden layer.

The input is $x_1, x_2, ..., x_d$. The weights from the input to the hidden layer are $w_{11}, w_{12}, ..., w_{21}, w_{22}, ..., w_{dk}$. The weights from the hidden layer to the output layer are $h_{11}, h_{12}, ..., h_{km}$. The activation of the hidden unit j $(1 < j < k)$ will be

$$ah_j = x_1 \times w_{1j} + x_2 \times w_{2j} + \cdots + x_m \times w_{mj}$$

The output of the jth hidden node will be $oh_j = f(ah_j)$. Activation of the output node l will be

$$a_1 = oh_1 \times h_{11} + oh_2 \times h_{21} + \cdots + oh_k \times h_{k1}$$

The output of the output node o_i will be $o_i = f(a_i)$. If the target output is $t_1, t_2, \cdots, t_n$, then the error at the output node o_i will be $e_i = t_i - o_i$. The weights between the hidden and output units is updated as follows:

$$h_{ji} = h_{ji} + \alpha \times o_j \times e_i \times g'(a_i)$$

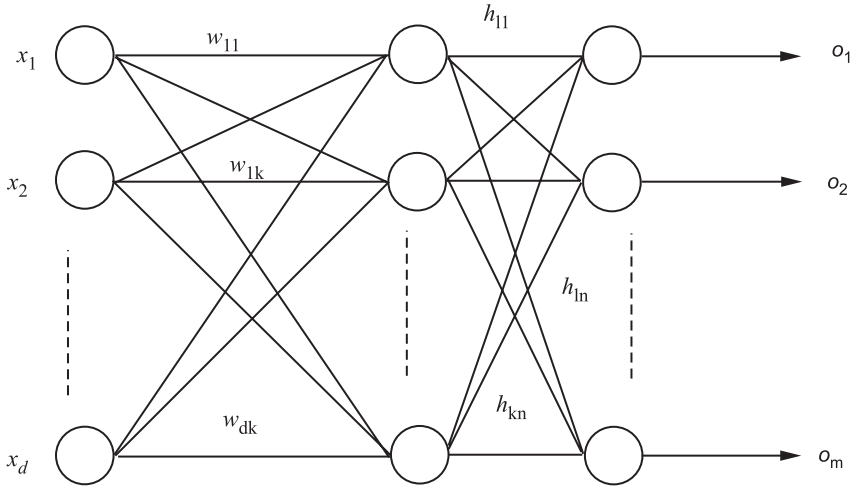


Figure 7.6 Multilayer perceptron with a hidden layer

Such an updation scheme is based on a gradient descent approach to minimise the error between the target and the obtained output values. The weights between the input units and the hidden units can then be updated in a similar manner. We require an activation function $g()$ that is differentiable. Note that the threshold function is not differentiable. A popular activation function used and differentiable is the sigmoid function shown in Figure 7.7.

This function can be written as $o(a) = \frac{1}{1+e^{-a}}$.

We show in Figure 7.8 a multilayer network with 2 input nodes, 2 hidden nodes and an output node.

It is possible to verify that this network can be used to characterise the non-linear decision boundary required to handle the exclusive OR (or XOR) function.

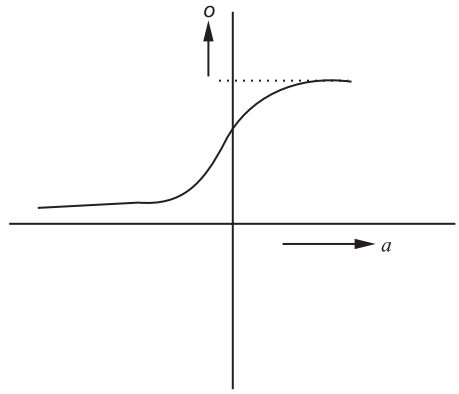


Figure 7.7 The sigmoid function

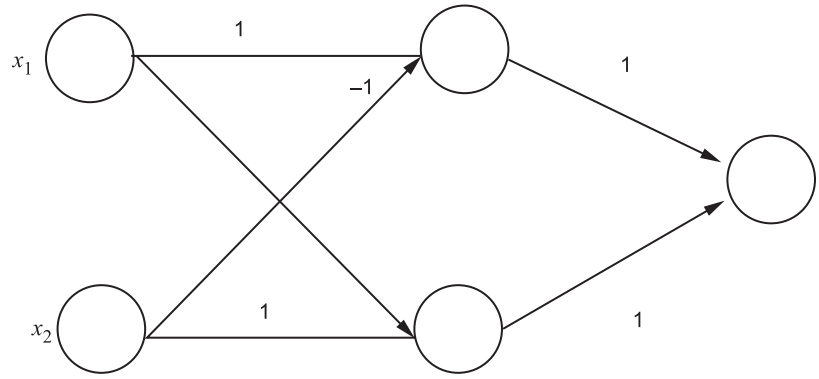


Figure 7.8 Multilayer network for XOR

7.4 SVM for Classification

Support vector machine (SVM) generates an abstraction in the form of a set of vectors called support vectors which belong to the training set of vectors. This may be illustrated with the help of the following example.

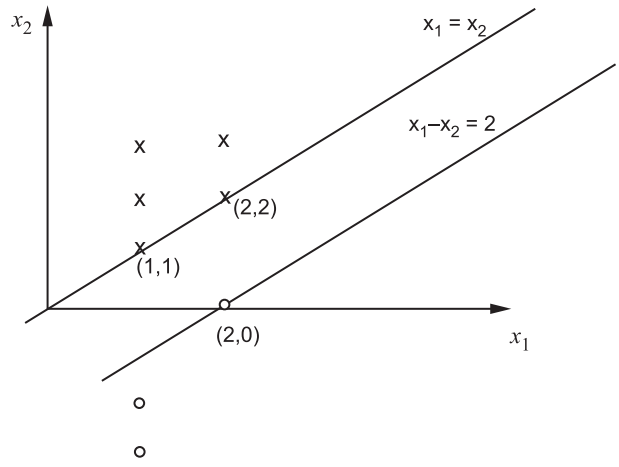


Figure 7.9 Illustration of support vector machines

EXAMPLE 14

Consider the three points in the two-dimensional space shown in Figure 7.9. These are drawn from two classes, X and O. Here, $(1,1)^t$ and $(2,2)^t$ are from class X and $(2,0)^t$ is from class O. The lines $x_1 - x_2 = 0$ and $x_1 - x_2 = 2$ characterise the boundaries of the classes X and O respectively. These lines are called *support lines* and the points are called *support vectors*. These three support vectors are adequate to characterise the classifier. Now consider adding points $(1,2)^t$, $(1,3)^t$ and $(2,3)^t$ from class X and $(2,-1)^t$, $(1,-2)^t$, and $(1,-3)^t$ from class O. They are properly classified using the support lines (in turn using the support vectors). In fact, any point of class X satisfying the property $x_1 - x_2 < 0$ is correctly classified. Similarly, any pattern from class O satisfying the property $x_1 - x_2 > 2$ is also correctly classified. The region between the two lines is called the margin and because the support lines correspond to the maximum value of the margin, they are as far away from each other as possible.

The line $x_1 - x_2 = 1$ is equidistant from the two decision lines and it forms the right choice for the decision boundary between the two classes. Points satisfying the property that $x_1 - x_2 < 1$ are classified as members of class X and those satisfying the condition $x_1 - x_2 > 1$ are of class O.

From the above discussion, we make the following observations.

1. SVM may be viewed as a binary (two class) classifier. It abstracts a linear decision boundary from the data and uses it to classify patterns belonging to the two classes.

2. Support vectors are some of the vectors falling on the support planes in a d-dimensional space. In two-dimensional space ($d = 2$), we have support lines which characterise the decision boundary.

3. SVMs learn, from the data, linear discriminants of the form $w^t + b$, where w is the weight vector and b is the threshold value. Unlike other learning algorithms (including perceptrons), SVMs learn the linear discriminant that maximises the margin.

4. When the two classes are linearly separable, it is easy to perceive and compute the maximum margin. So, we start our discussion with linearly separable classes.

7.4.1 Linearly Separable Case

We begin with a simple two-dimensional example.

EXAMPLE 15

Consider again the points $(1, 1)^t$ and $(2, 2)^t$ from class X and $(2, 0)^t$ from class O shown in Figure 7.9. They are linearly separable. In fact, we can draw several (possibly infinite) lines correctly separating the two classes represented by these three points. Using the perceptron learning algorithm, we get the decision boundary characterised by $x_1 - 3x_2 - 1 = 0$ (refer to Exercise 20). Two other possible separating lines are $x_1 - 2x_2 = 0$ and $x_1 - x_2 = 1$. We may put additional constraints to fix the line characterising the decision boundary between the two classes.

In the case of the SVM, we choose that line which provides maximum margin as a decision boundary. When the patterns are linearly separable, we can get the linear decision boundary corresponding to the maximum margin solution. For example, consider the three patterns in Figure 7.9. In this two-dimensional example, two points of a class are adequate to characterise the support line. So, for two points $(1, 1)^t$ and $(2, 2)^t$ from class X, the support line is $x_1 - x_2 = 0$. Now consider a line parallel to this support line and passing through the point $(2, 0)^t$ from class O; this is $x_1 - x_2 = 2$ and it is the support line for class O. These two lines, namely $x_1 - x_2 = 0$ and $x_1 - x_2 = 2$ characterise the margin. So, the decision boundary which is equidistant from these two lines is characterised by $x_1 - x_2 = 1$.

It is possible to derive the linear classifier of the form $f(x) = w^{tx} + b$ as follows. Consider two points $(1, 0)^t$ and $(2, 1)^t$ on the line $x_1 - x_2 = 1$. Because these points are on the decision boundary, we need $w = (w_1, w_2)^t$ to satisfy

$w_1 + b = 0$ (corresponding to $(1, 0)^t$) and

$2w_1 + w_2 + b = 0$ (for the point $(2, 1)^t$)

From these two equations, we get $w_1 + w_2 = 0$; so, w is of the form $(a, -a)^t$ and correspondingly, $b = -a$. In general, a is any constant. However, it is good to choose a in a normalised manner. For example, it is convenient to choose a such that

$w^t x + b = 0$ for all the points on the decision boundary

$w^t x + b = 1$ for all the points on the support line $x_1 - x_2 = 2$

$w^t x + b = -1$ for all the points on the support line $x_1 - x_2 = 0$

This can be achieved by choosing a value of 1 for a. So, correspondingly

$w = (1, -1)^t$ and $b = -1$

Let us consider computation of the normal distance between any point on the hyperplane $w^t x + b = 1$ and the decision boundary given by the plane $w^t x + b = 0$ in a d-dimensional space. Using equation (7.5), we know that the distance is given by $\frac{w^t x + b}{||w||}$. But, because of the location of x, we have $w^t x + b = 1$. So, the distance between the two planes

$w^t x + b = 1$ and $w^t x + b = 0$ is $\frac{1}{|| w ||}$

Similarly, the distance between the planes $w^t x + b = -1$ and $w^t x + b = 0$ is also $\frac{1}{||w||}$. So, the distance between the two support planes, $w^t x + b = 1$ and $w^t x + b = -1$ is $\frac{2}{||w||}$. This is called the margin. We illustrate it using the following example.

EXAMPLE 16

Consider the linearly separable problem considered in Example 13. In such a case, the distance between the two support lines is given by $\frac{2}{||w||}$, where $w = (1, -1)$. So, the margin in this specific example is $\sqrt{2}$.

It is possible to maximise the distance (margin) by minimising a monotone function of $||w||$. Further, each pattern imposes a constraint on the value of w because for each pattern x from class X, we want $w^t x + b \leq -1$ and for every pattern from the positive class (class O), we need $w^t x + b \geq 1$. A popular formulation of the optimisation problem is

$$\text{Minimise} \quad \frac{|| w ||^2}{2} \tag{7.16}$$

$$\text{such that} \quad w^t x + b \leq -1 \ \forall x \in X \text{ and} \tag{7.17}$$

$$w^t x + b \geq 1 \ \forall x \in \text{O} \tag{7.18}$$

We illustrate the solution to the linearly separable problem using an example.

EXAMPLE 17

Consider a one-dimensional data set of 3 points shown in Table 7.17.

Table 7.17 SVM on one-dimensional data

Pattern No.	x	Class
1	1	X
2	2	X
3	4	O

Here, the required w is a scalar and so the criterion function to be minimised is $\frac{w^2}{2}$ and the three constraints are (one per pattern):

$w + b \leq -1$, $2w + b \leq -1$, and $4w + b \geq 1$.

Such a constrained optimisation problem is solved by forming the Lagrangian given by

$$\text{Min } J(w) = \frac{w^2}{2} - \alpha_1(-w - b - 1) - \alpha_2(-2w - b - 1) - \alpha_3(4w + b - 1) \tag{7.19}$$

where αs are the Lagrange variables, one per each constraint. By differentiating with respect to w and b and equating the partial derivatives to 0, we get

$$\frac{\delta J}{\delta w} = 0 \Rightarrow w = -\alpha_1 - 2\alpha_2 + 4\alpha_3 \tag{7.20}$$

$$\frac{\delta J}{\delta b} = 0 \Rightarrow 0 = \alpha_1 + \alpha_2 - \alpha_3 \tag{7.21}$$

Similarly, by differentiating with αs, we get

$$-w - b - 1 = 0 \tag{7.22}$$

$$-2w - b - 1 = 0 \tag{7.23}$$

$$4w + b - 1 = 0 \tag{7.24}$$

Note that equations (7.22) and (7.23) contradict each other. So, using equations (7.23) and (7.24), we get $w = 1$ and $b = -3$. Also, for the optimal solution, we require either $\alpha_1 = 0$ or $-w - b - 1 = 0$. For the values selected ($w = 1$ and $b = -3$), $-w - b - 1 \neq 0$; so, $\alpha_1 = 0$. From equation (7.21), we get $\alpha_2 = \alpha_3$. Now from equation (7.20), we get $1 = 2\alpha_3$. So, $\alpha_2 = \alpha_3 = \frac{1}{2}$. Note that the decision boundary is specified by $wx + b = 0$. In this case, $w = 1$ and $b = -3$. So, the decision boundary here is $x - 3 = 0$ or $x = 3$.

Even though the data in the above example is one-dimensional, the solution procedure is general enough. We illustrate it with a two-dimensional example.

EXAMPLE 18

Consider the three two-dimensional data points shown in Example 13 where (1, 1), and (2, 2) are from class X and (2, 0) is from O. Here w is a two-dimensional vector and so the objective function and the three constraints are

$$\text{Minimise} \quad \frac{||\ w\ ||^2}{2} \tag{7.25}$$

$$\text{s. t.} \quad w^t x + b \leq -1 \ \ \forall\ x \text{ in X and} \tag{7.26}$$

$$w^t x + b \geq 1 \ \ \forall \quad x \ \ \in \ O \tag{7.27}$$

Note that the first constraint is $w^t x + b \leq -1$, where $x = (1, 1)^t$. So, we get equivalently $w_1 + w_2 + b \leq -1$. By writing the constraints in terms of the two components of w, namely w_1 and w_2 and the corresponding components of x, we have the Lagrangian to be

$$J(w) = \frac{||w||^2}{2} - \alpha_1(-w_1 - w_2 - b - 1) - \alpha_2(-2w_1 - 2w_2 - b - 1) - \alpha_3(2w_1 + b - 1) \tag{7.28}$$

Note that by differentiating $J(w)$ with respect to w and equating to 0, we get

$$w + \alpha_1(1, 1)^t + \alpha_2(2, 2)^t - \alpha_3(2, 0)^t = 0 \tag{7.29}$$

The above equation leads to

$$w_1 = -\alpha_1 - 2\alpha_2 + 2\alpha_3 \tag{7.30}$$

$$w_2 = -\alpha_1 - 2\alpha_2 \tag{7.31}$$

By differentiating $J(w)$ with respect to b and equating to 0, we have

$$\alpha_1 + \alpha_2 - \alpha_3 = 0 \tag{7.32}$$

Also, by differentiating with respect to αs and equating to 0, we get

$$-w_1 - w_2 - b - 1 = 0 \tag{7.33}$$

$$-2w_1 - 2w_2 - b - 1 = 0 \tag{7.34}$$

$$2w_1 + b - 1 = 0 \tag{7.35}$$

From equations (7.34) and (7.35), we get $-2w_2 - 2 = 0$ which implies that $w_2 = -1$. Further, from equations (7.33) and (7.35), we get $w_1 + w_2 = 0$ which means $w_1 = 1$. With these values and equation (7.35), we get $b = -1$. So, the solution is given by $b = -1$ and

$$w = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Note that in this case, all the three vectors are support vectors. Using equations (7.30), (7.31), and (7.32), it is possible to solve for αs. The solution is $\alpha_1 = 1$, $\alpha_2 = 0$, and $\alpha_3 = 1$. It is also possible to show in general that

$$w = \sum_i \alpha_i y_i x_i \tag{7.36}$$

where y_i is $+1$ for positive patterns (patterns from O) and -1 for negative patterns (patterns from X).

7.4.2 Non-linearly Separable Case

In the previous sub-section, we have seen a mechanism for dealing with classification of patterns when the classes are linearly separable. However, it is possible that the classes may not be linearly separable in some cases. We illustrate with a simple example.

EXAMPLE 19

Consider a one-dimensional collection of 4 labelled patterns: -3, and 3 from class O and -1 and 1 from class X. They are not linearly separable. However, if we can map these points using a function of the form $f(x) = x^2$, we get 9 and 9 for patterns in O and 1 and 1 for patterns in X and these modified patterns are linearly separable.

Another possibility is to convert the one-dimensional patterns into two-dimensional patterns. This is given in the following example.

EXAMPLE 20

$f(x) = (x, x^2)^t$ maps the points of class O to $(-3, 9)^t$ and $(3, 9)^t$ and points of class X to $(-1, 1)^t$ and $(1, 1)^t$. These are also linearly separable.

We define the dot product between two vectors $(p_1, p_2)^t$ and $(q_1, q_2)^t$ as $p_1q_1 + p_2q_2$. In general, it is possible to map points in a d-dimensional space to some D-dimensional space ($D > d$) to explore the possibility of linear separability. This may be illustrated using the following example.

EXAMPLE 21

Consider the function f shown in Table 7.18.

Table 7.18 Truth table for $f(x_1, x_2)$

x_1	x_2	$f(x_1, x_2)$
0	0	1
0	1	0
1	0	0
1	1	1

If we consider the output values 0 and 1 to represent classes X and O respectively, then these two classes are not linearly separable. This may be shown using a simple argument. Let us consider, on the contrary, that the two classes are linearly separable. This means that there exists a line of the form $\alpha_1 x_1 + \alpha_2 x_2 + c = 0$ which separates points of class X from those of class O. More specifically, a point (x_1, x_2) labelled X will satisfy the inequality $\alpha_1 x_1 + \alpha_2 x_2 + c < 0$ and similarly a point (x_1, x_2) labelled 0

will satisfy the inequality $\alpha_1 x_1 + \alpha_2 x_2 + c > 0$. Note that the first and the fourth rows in the table correspond to class 0 and the remaining two rows (second and the third) correspond to class X. So, we get the following inequalities:

From the first row: $c > 0$

From the fourth row: $\alpha_1 + \alpha_2 + c > 0$

From the second row: $\alpha_2 + c < 0$ and

From the third row: $\alpha_1 + c < 0$.

These four inequalities lead to a contradiction as by adding the first two inequalities, we get

$$\alpha_1 + \alpha_2 + 2c > 0 \tag{7.37}$$

and addition of the last two inequalities gives us

$$\alpha_1 + \alpha_2 + 2c < 0 \tag{7.38}$$

contradicting it. This clearly shows that the two classes are not linearly separable. However, by using the feature $x_1 \wedge x_2$ also, we get a higher dimensional representation (the additional feature makes the 2-dimensional space a 3-dimensional space). The corresponding truth table of f is shown in Table 7.19.

Table 7.19 Truth table for $f(x_1, x_2, x_1 \wedge x_2)$

x_1	x_2	$x_1 \wedge x_2$	$f(x_1, x_2, x_1 \wedge x_2)$
0	0	0	1
0	1	0	0
1	0	0	0
1	1	1	1

Note that in three-dimensional space, the classes are linearly separable. In this case, we need to consider a plane that separates points of class 0 from those of class X (refer to Exercise 7.23).

In addition, in SVMs it is done such that the dot product computations in the D-dimensional space are carried out by using the dot product computations in the d-dimensional space. For example, consider mapping from a 2-dimensional space to a 3-dimensional space such that a point $(p, q)^t$ in 2-d space is mapped to $(p^2, q^2, \sqrt{2}pq)^t$ in 3-d space. Now the dot product between the two points in 3-d space can be characterised using a function of the dot product of the corresponding vectors in 2-d space (refer to Exercise 22).

Discussion

The focus of this chapter was classification of multi-dimensional patterns using a linear decision boundary. Specifically, the perceptron classifier, an algorithm for learning the weight vector of the classifier was discussed in detail. There is a great amount of work on related classifiers and extensions leading to neural networks in literature. Both feed-forward networks and multilayer perceptrons were discussed. SVMs are also linear decision boundary based classifiers possibly in a higher dimensional space. They have received considerable attention in the past decade. A brief introduction to SVMs is provided in this chapter.

Further Reading

Duda et al's book (2001) on pattern classification is an excellent introduction to linear discriminant functions and generalised linear discriminant functions.

Minsky and Papert (1988) give an authoritative discussion on perceptrons which clearly establishes the power and limitations of perceptron networks. Rumelhart et al. (1986) introduce back-propagation algorithms which are popular in training multilayer feed-forward networks. Freeman and Skapura (1992) gives an excellent introduction to a variety of neural networks and the corresponding training algorithms.

Back-propagation algorithm for training multilayer perceptron is inherently a gradient descent approach. It was observed that using this algorithm, the perceptron gets attracted to the locally optimal solution of the optimisation problem. This motivated researchers to look for more robust schemes for training neural networks. SVMs may be viewed as state-of-the-art linear discriminant functions. Burges (1998) provides an excellent and authoritative tutorial coverage of SVMs and their role in pattern recognition. Platt (1998) describes an efficient scheme to train the SVM.

Exercises

1. Show that the patterns in Figure 7.1 corresponding to class X satisfy the property $x_1 - x_2 < 0$ (they are in the negative half space) and patterns labelled O are in the positive half space.
2. Consider the two-dimensional labelled patterns shown in Figure 7.1. Show that the lines $x_1 = x_2$, $2x_1 - 2x_2 = -1$ and $2x_1 - 2x_2 = 1$ are parallel to each other. What can you say about the w and b in each of these cases?
3. Show that the weight vector w is orthogonal to the decision boundary in d-dimensional space, where $d \geq 2$.

4. Let θ be the angle between w and x. Show that the value of $\cos\ \theta$ is positive if $w^t x\ > 0$.

5. Consider the two-dimensional labelled patterns shown in Figure 7.1 and the decision boundary given by $x_1 - x_2 = 1$. Show that the distance from any point of the form (α, α), where α is a real number, to the decision boundary is $\frac{-1}{\sqrt{2}}$.

6. Consider the truth table corresponding to XOR given in Table 7.20. Consider output 1 to be class X and 0 to be class O. Show that the classes are not linearly separable.

Table 7.20 Truth table for XOR

x	y	XOR (x, y)
0	0	0
0	1	1
1	0	1
1	1	0

7. Consider the two-dimensional collection of patterns from two classes X and O shown in Table 7.21. Use the perceptron learning algorithm to get the corresponding w and b values. How do you classify the pattern $(1, 1)^t$ using the classifier?

Figure 7.21 Description of the patterns

Pattern No.	1	2	Class
1	0.5	1.0	X
2	1	2	X
3	0.5	1.5	X
4	1.5	1.5	X
5	1.5	2	X
6	4.5	1.5	O
7	5	1.5	O
8	4.5	2.5	O
9	5.5	1.5	O

8. Consider the three patterns $(1, 1)^t$ and $(2, 2)^t$ from class X and $(2, 0)^t$ from class O. Use the perceptron learning algorithm to show that the decision boundary is $x - 3y - 1 = 0$.

9. Consider the two class problem of separating "X" from "*" discussed in Section

7.2.2. Show that the training algorithm obtains the weight vector

$$w_4 = \begin{pmatrix} -9 \\ 5 \\ 1 \end{pmatrix}$$

that classifies all the 6 patterns in Table 7.6 correctly.

10. Consider the two-class problem of separating "O" from "*" discussed in Section 7.2.2. Show how the training algorithm obtains the weight vector

$$w_{o*} = \begin{pmatrix} -4 \\ 5.5 \\ 0 \end{pmatrix}$$

that classifies all the 6 patterns in Table 7.7 correctly.

11. Consider classification of patterns in Table 7.4 using 3 binary classifiers as discussed in Section 7.2.2. Here, each classifier deals with a pair of classes. Three of the 9 patterns in the table were classified (x_1, x_5, and x_9). Show that all the remaining 6 patterns in the table can be classified using the weight vectors w_{xo}, w_{x*}, and w_{o*}.

12. Consider the three class classification problem given in Exercise 9. Can any two-dimensional pattern be properly classified? Specifically, using the weight vectors obtained, can you classify the pattern (2, 0.5)?

13. Consider the data given in Table 7.9. Show that w_o is

$$w_o = \begin{pmatrix} -1.5 \\ 5.5 \\ -21 \end{pmatrix}$$

14. Consider the data given in Table 7.10. Show that w_* is

$$w_o = \begin{pmatrix} 2.5 \\ -11.5 \\ -2 \end{pmatrix}$$

15. Consider the multi-class classification problem discussed in Section 7.2.2, where the 3 classifiers were one versus the rest. Using the data set shown in Table 7.4, the vectors w_x, w_o, and w_* were obtained and used in classifying 3 patterns. Use the weight vectors to classify the remaining 6 patterns, x_2, x_3, x_4, x_6, x_7, and x_8.

16. Show that the data given in Table 7.12 is the transformed and normalised version of the data given in Table 7.11 using the details provided in Example 11.

17. Show that for the Boolean OR problem discussed in Example 11, the sequence of weight vectors which classifies all the four patterns correctly reaches w_{10}.

18. Complete the missing details in the training process described in Section 7.3.2 based on the data given in Table 7.15.

19. Consider the multilayer perceptron shown in Figure 7.8. Show that it can be used to describe the XOR function using the discussion in the section on multilayer networks.

20. Consider the two class problem given in Example 14. Here, (1,1) and (2,2) are from class X and (2,0) is from class O. Show that the perceptron training algorithm gives the decision boundary $x - 1 - 3x_2 - 1 = 0$.

21. Consider the data set shown in Figure 7.9. Use the SVM learning algorithm discussed in Section 7.3.1 to compute w, b, and α_i for $i = 1, \cdots, 9$.

22. Consider the mapping shown in Section 3.2 which maps two-dimensional points of the form (p, q) to three-dimensional points of the form $(p^2, q^2, \sqrt{2}pq)$. How do you specify the dot product of two vectors in three-dimensional space in terms of the dot products of the corresponding points in two-dimensional space.

23. Consider truth table of f shown in Table 7.19. Let output 0 correspond to class X and 1 correspond to class 0. Show that the two classes are linearly separable.

Computer Exercises

1. Develop a program for learning perceptron weight vectors that can be used to classify patterns belonging to two linearly separable classes. Use it to classify the two class data shown in Table 7.3.

2. Extend the program developed in Computer Exercise 1 to handle multi-class classification problems. Use it to classify the three class data shown in Table 7.4.

3. Develop a program for training a feed-forward network.

4. LIBSVM (http://www.csie.ntu.edu.tw/ cjlin/libsvm/) and SVM^{light} (http://svmlight.joachims.org/) are two popular packages available for training SVMs. Familiarise yourself with these packages and solve some two class classification problems.

Bibliography

1. Duda, R. O., P. E. Hart, and D. G. Stork. *Pattern Classification.* Second Edition. Wiley-Interscience. 2001.

2. Minsky, M. L. and S. Papert. *Perceptrons: An Introduction to Computational Geometry.* Cambridge: MIT Press. 1988.

3. Rumelhart, D. E., G. E. Hinton, and R. J. Williams. Learning internal representations by backpropagating errors. *Nature* 323(99): 533–536. 1986.

4. Burges, C. J. C. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery* 2: 121–168. 1998.

5. Platt, J. C. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods-Support Vector Learning* edited by B. Scholkopf, C. J. Burges, and A. Smola. pp. 185–208. MIT Press. 1998.

6. Freeman, J. A. and D. M. Skapura. Neural Networks. Pearson Education. 1992.

8 Combination of Classifiers

Learning Objectives

At the end of this chapter you will

- Know why an ensemble of classifiers are more often used than just one classifier. There are different methods for constructing ensembles. They are
  - Sub-sampling the training examples using
    - Bagging
    - Leaving out disjoint sub-sets
    - ADABOOST
  - Manipulating input features
  - Manipulating output targets
  - Injecting randomness into the learning algorithm
- Learn the different methods for combining classifiers

8.1 Introduction

A combination or an ensemble of classifiers is a set of classifiers whose individual decisions are combined to classify new examples. A combination of classifiers is often much more accurate than the individual classifiers that make them up. One reason for this could be that the training data may not provide sufficient information for choosing a single best classifier and a combination is the best compromise. Another reason could be that the learning algorithms used may not be able to solve the difficult search problem posed. Since solving the search problem maybe difficult, suitable heuristics may be used in the search. As a consequence of this, even though with the training examples and prior knowledge, a unique best hypothesis exists, we may not be able to find it. A combination of classifiers is a way of compensating for imperfect classifiers. The learning algorithms we use may give us good approximations to the true value but may not be the right hypothesis. By taking a weighted combination of these approximations, we may be able to represent

the true hypothesis. In fact, the combination could be equivalent to very complex decision trees.

EXAMPLE 1

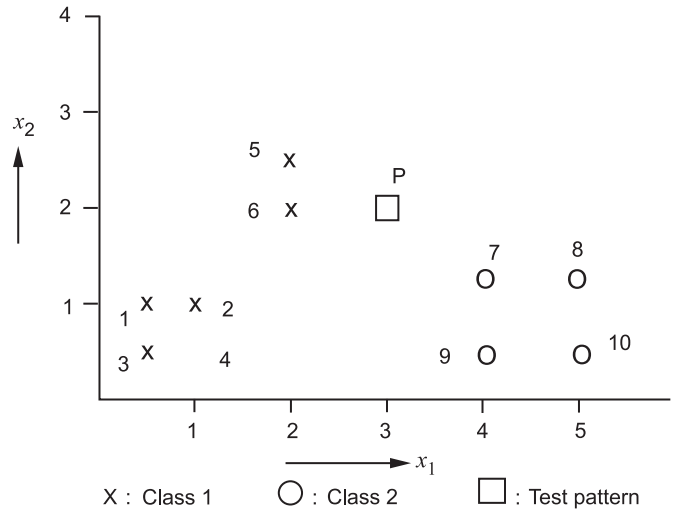


Figure 8.1 Training data set and the test pattern

Figure 8.1 shows a data set consisting of two classes, the class X and the class O. The patterns are :

$$\mathrm{X}_1 = (0.5, 1, \mathrm{X}); \quad \mathrm{X}_2 = ((1, 1, \mathrm{X}); \quad \mathrm{X}_3 = (0.5, 0.5, \mathrm{X}); \quad \mathrm{X}_4 = (1, 0.5, \mathrm{X});$$

$$\mathrm{X}_5 = (2, 2.5, \mathrm{X}); \quad \mathrm{X}_6 = (2, 2, \mathrm{X}); \quad \mathrm{X}_7 = (4, 1.25, \mathrm{O}); \quad \mathrm{X}_8 = (5, 1.25, \mathrm{O});$$

$$\mathrm{X}_9 = (4, 0.5, \mathrm{O}); \quad \mathrm{X}_{10} = (5, 0.5, \mathrm{O});$$

If different classifiers are formed by taking a different sub-set of the training set, any training set which contains either or both the patterns X_5 and X_6 will classify the test pattern P at (3, 2) as belonging to Class X using the nearest neighbour algorithm. But if these patterns are not in the sub-set, then P is classified as belonging to Class O. If the majority of the classifiers do not have either X_5 or X_6, if the combination of the classifiers is done according to a majority vote, P will be classified as belonging to Class O. If a majority of the classifiers contain either X_5 or X_6 or both, then P will be classified as belonging to Class X.

8.2 Methods for Constructing Ensembles of Classifiers

The methods generally vary the input given to each classifier. These include sub-sampling the training set, manipulating the features of the training set, manipulating the output target, injecting randomness and some methods which are specific to particular algorithms.

8.2.1 Sub-sampling the Training Examples

The learning algorithm is run several times, each time with a reduced data set obtained from the training set. This method runs particularly well on algorithms where the output classifier undergoes major changes in response to small changes in the training set. The data set can be reduced by choosing a sub-set of the training set or by using a different sub-set of the features in each case.

Bagging

On each run, this technique presents the classifier with a training set that consists of m training examples drawn randomly from the original training set of n items. Such a training set is called a bootstrap replicate of the original training set and the technique is called *bootstrap aggregation*. Each bootstrap replicate contains about two-thirds of the original training set with some patterns possibly appearing several times.

EXAMPLE 2

Consider the following training set :

$$\mathrm{X}_1 = (1, 1, \mathrm{X}); \quad \mathrm{X}_2 = (2, 1, \mathrm{X}); \quad \mathrm{X}_3 = (3.3, 1, \mathrm{X}); \quad \mathrm{X}_4 = (1, 2, \mathrm{X});$$

$$\mathrm{X}_5 = (2, 2, \mathrm{X}); \quad \mathrm{X}_6 = (5, 1, \mathrm{O}); \quad \mathrm{X}_7 = (6, 1, \mathrm{O}); \quad \mathrm{X}_8 = (5, 2, \mathrm{O});$$

$$\mathrm{X}_9 = (6, 2, \mathrm{O}); \quad \mathrm{X}_{10} = (5, 3, \mathrm{O})$$

Here each triplet gives the first feature, the second feature and the class label. This is shown in Figure 8.2.

If there is a test pattern at (4, 2), using the given training set, it is closer to Class O and will be classified in this case as belonging to Class O if the nearest neighbour algorithm is used. If bagging is used and two patterns are drawn at random from each class to be used for classification, the class of the test pattern will vary depending on the patterns drawn. If patterns 3 and 5 are drawn from Class X and patterns 7 and 9 are drawn from Class O, then the test pattern will be classified as belonging to Class X. If patterns 1 and 4 are drawn from Class X and patterns 6 and 7 from Class O, then the test pattern will be classified as belonging to Class O. If the classification is

done a number of times with different patterns from the two classes at random, these results have to be combined to find the class of the test pattern.

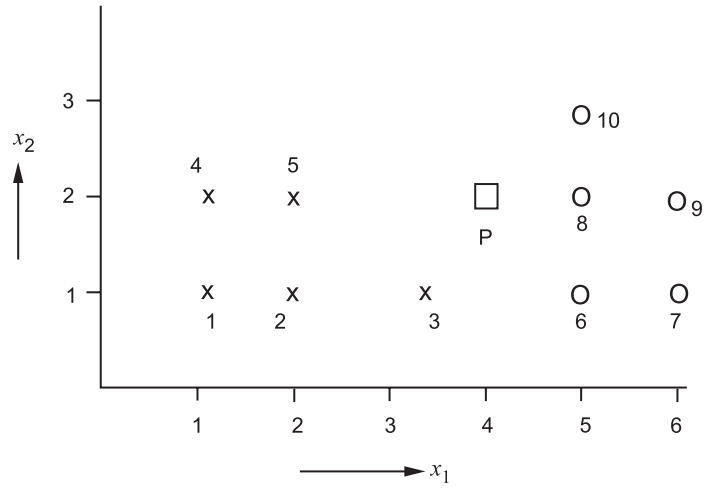


Figure 8.2 Training data set having two classes

Leaving Out Disjoint Sub-sets

The original training set is divided into a number of disjoint sub-sets. Then different overlapping training sets are constructed by dropping one of these sub-sets.

EXAMPLE 3

Consider the data set given in Figure 8.2. Let us take the disjoint sub-sets as $S_1 = \{1, 2\}$, $S_2 = \{4, 5\}$, $S_3 = \{3\}$, $S_4 = \{6, 7\}$, $S_5 = \{8, 10\}$, $S_6 = \{9\}$.

If sub-sets S_1 and S_4 are left out, then the data set will consist of the points {3, 4, 5, 8, 9, 10} and P will be classified as belonging to Class O. In this way, by leaving out one sub-set from each class, we can get different sets of points. If S_1 and S_5 are left out, the data set will consist of the points {3, 4, 5, 6, 7, 9} and P will be classified as belonging to Class X. If S_1 and S_6 are left out, the data set will consist of {3, 4, 5, 6, 7, 8, 10} and P will be classified as belonging to Class O. If S_2 and S_4 are left out, the data set will consist of {1, 2, 3, 8, 9, 10} and P will be classified as belonging to Class O. If S_2 and S_5 are left out, the data set will consist of {1, 2, 3, 6, 7, 9} and P will be classified as belonging to Class X. If S_2 and S_6 are left out, the data set will be {1, 2, 3, 6, 7, 8, 10} and P will be classified as belonging to Class O. If S_3 and S_4 are left out, the data set will consist of {1,2,4,5,8,9,10} and P will be classified as belonging to Class O. If S_3 and S_5 are left out, the data set will consist of {1, 2, 4, 5,

6, 7, 9} and P will be classified as belonging to Class X. If S_3 and S_6 are left out, the data set will consist of {1, 2, 4, 5, 6, 7, 8, 10}, and P will be classified as belonging to Class O. Thus, if this is done a number of times, the classification of P will depend on which sub-sets are left out. The combination of these decisions will decide on the classification of P.

ADABOOST Algorithm

The general problem of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb is referred to as *boosting*. The booster is provided with a set of labelled training examples $(x_1, \theta_1), ..., (x_N, \theta_N)$, where θ_i is the label associated with instance x_i. On each round $t = 1, ..., T$, the booster devices a distribution D_t over the set of examples and uses a weak hypothesis h_t having low error ϵ_t with respect to D_t. Any classifier can be used at this point. The weak classifier can be a decision stump—a decision tree with depth one. In other words, the classification is based on a single decision node. Thus, distribution D_t specifies the relative importance of each example for the current round. After T rounds, the booster must combine the weak hypothesis into a single prediction rule.

One boosting algorithm is the ADABOOST which maintains a probability distribution $p_t(x)$ over the training set. In each iteration t, it draws a training set of size m by sampling with replacement according to the probability distribution $p_t(x)$. The classifier is used on this training set. The error rate e_t of this classifier is computed and used to adjust the probability distribution on the training set. The probability distribution is obtained by normalising a set of weights, $w_t(i), i = 1, .., n$ over the training set. The effect of the change in weights is to place more importance on training examples that were misclassified by the classifier and less weight on the examples that were correctly classified. The final classification is constructed by a weighted vote of the individual classifiers. Each classifier is weighted according to its accuracy for the distribution p_t that it was trained on.

If the input is a training set S of size n, the inducer is $\mathcal{I}$ and the number of trials T, the algorithm is as follows :

STEP 1: $S' = S$ with weights assigned to be 1; $m = n$;

STEP 2: Consider $i = 1$ to T

STEP 3: $C_i = \mathcal{I}\ (S')$

STEP 4: $\epsilon_i = \dfrac{1}{m} \displaystyle\sum_{x_j \in S' : C_i(x_j) \neq y_j} \text{weight}(x)$

STEP 5: If $\epsilon_i > \frac{1}{2}$, set S' to a bootstrap sample from S with weight 1 for every instance and go to Step 3.

STEP 6: $\beta_i = \frac{\epsilon_i}{(1-\epsilon_i)}$

STEP 7: For each $x_j \in S'$, if $C_i(x_j) = y_j$ then weight(x_j) = weight(x_j) . β_i

STEP 8: Normalise the weights of instances so that the total weight of S' is m.

STEP 9:

$$C^*(x) = \arg\max_{y \in Y} \sum_{i:C_i(x)=y} \log \frac{1}{\beta_i}$$

In the above algorithm, every weight is taken as 1 and we start with the whole training data. A classifier is chosen. The ϵ error of using this classifier is obtained by adding the weights of the patterns which are misclassified and dividing the sum by the total number of patterns m. This gives the equation in Step 4. β is calculated as in Step 6 which represents the error in classification. The weights of the samples which are classified correctly are updated(reduced) by multiplying their weight with β. They are then normalised so that they sum to m. This is carried out for different classifiers. The inducer (I) decides the classifier to be used. Step 5 is used to see that the error does not become too high for the classifier. Step 9 gives the equation to be used to combine the classifiers for a test pattern. The term $\log \frac{1}{\beta}$ represents the classification accuracy of the particular classifier. A test pattern P is classified according to different hypotheses. For each class, the summation of $\log \frac{1}{\beta}$ is carried out only for the hypothesis for which it is classified as belonging to that class. The class which has the highest term for the summation is chosen as the class of the test pattern.

EXAMPLE 4

Consider Figure 8.2. Let a weight of 1 be assigned to all the samples, i.e., weight(i)=1, i = 1,...,10. Consider three hypotheses where Hypothesis 1 and Hypothesis 2 are decision stumps.

Hypothesis 1

Let the first hypothesis be that if $x_1 \leq 3$, the pattern belongs to Class X and Class O otherwise. This hypothesis misclassifies pattern 3. Which means

$$\epsilon_1 = \frac{1}{10} = 0.1$$

$$\beta_1 = \frac{0.1}{0.9} = 0.1111$$

weight(1) = $1 \times 0.1111 = 0.1111$

Similarly the weights of the other patterns except pattern 3 will be 0.1111. Only the weight of pattern 3 remains as 1. Normalising,

$$\text{weight}(1) = \frac{0.1111}{1.9999} \times 10 = 0.5555$$

$$\text{weight}(2) = 0.5555, \text{weight}(4) = 0.5555, \text{weight}(5) = 0.5555,$$

$$\text{weight}(6) = 0.5555, \text{weight}(7) = 0.5555, \text{weight}(8) = 0.5555,$$

$$\text{weight}(9) = 0.5555 \text{ and weight}(10) = 0.5555$$

$$\text{weight}(3) = \frac{1}{1.9999} \times 10 = 5.0002$$

Hypothesis 2

Let the second hypothesis be that if $x_1 \leq 5$, the pattern belongs to Class X and Class O otherwise.

$$\epsilon_2 = \frac{1}{10} \times (0.5555 + 0.5555 + 0.5555) = 0.16665$$

$$\beta_2 = \frac{0.16665}{1 - 0.16665} = 0.2000$$

$$\text{weight}(1) = 0.5555 \times 0.2 = 0.1111; \text{weight}(2) = 0.1111;$$

$$\text{weight}(3) = 5.0002 \times 0.2 = 1.00004; \text{weight}(4) = 0.1111;$$

$$\text{weight}(5) = 0.1111; \text{weight}(6) = 0.5555;$$

$$\text{weight}(7) = 0.1111; \text{weight}(8) = 0.5555;$$

$$\text{weight}(9) = 0.1111; \text{weight}(10) = 0.5555;$$

Normalising,

$$\text{weight}(1) = \frac{0.1111}{3.33314} \times 10 = 0.333319; \text{weight}(2) = 0.333319;$$

$$\text{weight}(3) = \frac{1.00004}{3.33314} \times 10 = 3.0003; \text{weight}(4) = 0.333319;$$

$$\text{weight}(5) = 0.333319; \text{weight}(6) = \frac{0.5555}{3.33314} \times 10 = 1.6666;$$

$$\text{weight}(7) = 0.333319; \text{weight}(8) = 1.6666;$$

$\text{weight}(9) = 0.333319; \text{weight}(10) = 1.6666;$

Hypothesis 3

Let the third hypothesis be that if $x_1 + x_2 \leq 3.5$, the pattern belongs to Class X and Class O otherwise. This hypothesis misclassifies pattern 3 and 5.

$$\epsilon_3 = \frac{1}{10} \times (3.0003 + 0.333319) = 0.3334$$

$$\beta_3 = \frac{0.3334}{1 - 0.3334} = 0.5002$$

$\text{weight}(1) = 0.333319 \times 0.5002 = 0.16673$; $\text{weight}(2) = 0.16673$;

$\text{weight}(3) = 3.0003$; $\text{weight}(4) = 0.16673$;

$\text{weight}(5) = 0.333319$; $\text{weight}(6) = 1.6666 \times 0.5002 = 0.8336$;

$\text{weight}(7) = 0.16673$; $\text{weight}(8) = 0.8336$; $\text{weight}(9) = 0.16673$; $\text{weight}(10) = 0.8336$;

Normalising,

$\text{weight}(1) = \frac{0.16673}{6.668069} \times 10 = 0.2502$; $\text{weight}(2) = 0.2502$;

$\text{weight}(3) = \frac{3.0003}{6.668069} \times 10 = 4.4995$; $\text{weight}(4) = 0.2502$;

$\text{weight}(5) = \frac{0.333319}{6.668069} \times 10 = 0.4999$; $\text{weight}(6) = \frac{0.8336}{6.668069} \times 10 = 1.2501$;

$\text{weight}(7) = 0.2502$; $\text{weight}(8) = 1.2501$;

$\text{weight}(9) = 0.2502$; $\text{weight}(10) = 1.2501$;

If we take a test pattern (4, 2), according to the first hypothesis, it belongs to class O; according to the second hypothesis, it belongs to class X; and according to the third hypothesis, it belongs to class O. For Class X,

$$\Sigma \log \frac{1}{\beta_i} = \log \frac{1}{0.2} = 0.699$$

For Class O,

$$\Sigma \log \frac{1}{\beta_i} = \log \frac{1}{0.1111} + \log \frac{1}{0.5002} = 1.2551$$

P will be classified as belonging to Class O.

8.2.2 Manipulating the Input Features

This technique manipulates the set of input features that is available to the classifier. A sub-set of the features can be taken. If all the features are significant, this technique is not likely to work. It works only when the input features are highly redundant and can be grouped together depending on some domain knowledge. This grouping can then be used to form the feature sub-sets.

EXAMPLE 5

In the example given in Figure 8.2, if the test point is at (4, 2), it is closer to pattern 8 and will therefore belong to Class O. If only the first feature is considered, then the test point will be closer to pattern 3 and therefore will be classified as belonging to Class X. If patterns of different classes are distinctly separable with respect to a feature or a group of features, then using these features for classification is meaningful. In Figure 8.2, it is obvious that the patterns are not separable with respect to feature 2.

8.2.3 Manipulating the Output Targets

This technique is called *error correcting output coding*. If there are a large number of classes, the C classes are partitioned into two sub-sets A_l and B_l. The input data is then re-labelled so that any of the patterns whose class is in set A_l is given the derived label 0 and the data whose class is in set B_l are given the derived label 1. This re-labelled data is given to the classifier which constructs the classifier h_l. By generating different partitions A_l and B_l every time, we can get L classifiers $h_1, ..., h_L$. A test data point is classified using each of the classifiers. If $h_l(x) = 0$, then each class in A_l receives a vote. If $h_l(x) = 1$, then each class in B_l receives a vote. After each of the L classifiers has voted, the class with the highest number of votes is selected as the prediction of the combination. Classification depends on the number and size of the partitions.

EXAMPLE 6

Figure 8.3 shows a data set with four classes 1, 2, 3 and 4. Let the classes be partitioned into two sub-sets A and B with two classes in each sub-set. Table 8.1 gives the classes in the two sub-sets for each trial, the class of test sample P and the points given to each of the four classes. If a class falls in the sub-set to which P is classified, it is given a point 1 else it is given 0.

Table 8.1 Classification using error correcting output coding

A	B	Class	Class 1	Class 2	Class 3	Class 4
1, 2	3, 4	A	1	1	0	0
2, 3	1, 4	B	1	0	0	1
1, 3	2, 4	A	1	0	1	0

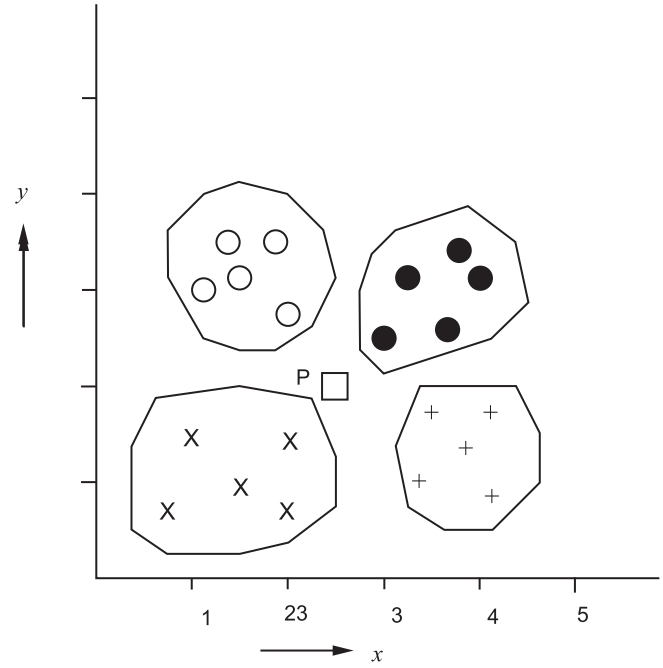


Figure 8.3 Training data set with four classes and the test pattern

From Table 8.1, Class 1 gets an overall score of 3 points; Class 2, 3 and 4 get a score of 1 point each. P is therefore classified as belonging to Class 1.

8.2.4 Injecting Randomness

This technique generates different classifiers by injecting randomness into the learning algorithm. In the decision tree algorithm, the order in which the features are chosen to make a decision at the internal nodes can be chosen randomly out of the top best tests. This change in the order of features chosen will result in different classifiers.

EXAMPLE 7

Let us consider Problem 1 from Chapter 6 where there are four coins a, b, c, and d. Three coins are of equal weight and one coin is heavier. It is necessary to find the heavier coin. The decision tree is given in Figure 6.1 and repeated here as Figure 8.4 for easy reference.

Now if the decision $a \geq b$ is taken first, then we can see that the decision tree will be different here. This is shown in Figure 8.5. By changing the order of the features chosen in the decision tree, we can see that it results in different classifiers.

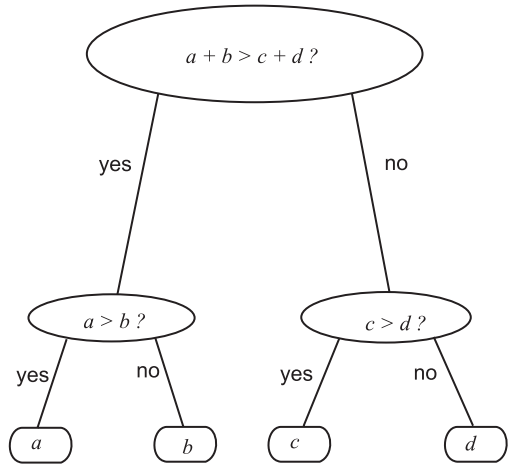


Figure 8.4 Decision tree for finding the counterfeit coin using $a + b \geq c + d$

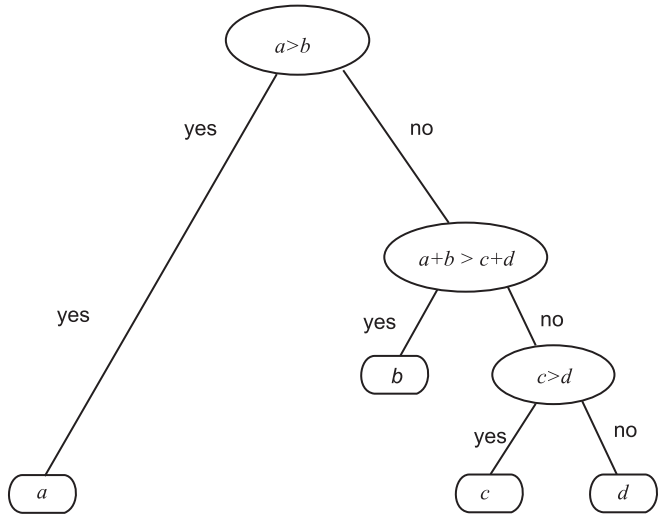


Figure 8.5 Decision tree for finding the counterfeit coin using $a \geq b$

Bootstrap sampling of the training data combined with injecting noise into the input features for the learning algorithm can also be used. One method draws training examples with replacement from the original training data, to train each member of a number of neural networks. The values of each training example are perturbed by adding Gaussian noise to the input features.

Another method of injecting randomness is the Markov chain Monte Carlo (MCMC) method which can be applied to neural networks as well as decision

trees. A Markov process is one for which the probability of a given future state, at any given moment, depends only on its present state, and not on any past states. A Markov process which is defined for a discrete set of time values is called a Markov chain. In the MCMC method, a Markov process is constructed to generate an infinite sequence of hypotheses h_l. A set of operators are defined that convert one h_l into another. In a decision tree, the operator might interchange a parent and child node in the tree or replace one node with another. The MCMC process uses this principle. It works by maintaining a current hypothesis h_l. At each step, it selects an operator, applies it to obtain h_{l+1} and then computes the likelihood of the resulting classifier on the training data. It then decides whether to keep h_{l+1} or discard it and go back to h_l. Usually the Markov process is run for a long time discarding all the generated classifiers and then collecting a set of L classifiers. These classifiers are combined using weighted vote according to their posterior probabilities.

EXAMPLE 8

Let us assume that we have a current hypothesis h_l with a probability of 0.6. If h_l has been generated by using a decision tree, let us generate hypothesis h_{l+1} by interchanging two adjacent nodes in the decision tree. Using hypotheses h_{l+1} if 8 patterns are classified correctly in the training set and 2 are misclassified, we can take the likelihood to be 0.8. If the likelihood is satisfactory, the hypothesis can be retained; otherwise it can be discarded.

8.3 Methods for Combining Classifiers

The simplest and most robust approach to combining classifiers is to take an unweighted vote. A refinement on the simple majority vote is appropriate when each classifier can produce class probability estimates rather than a simple classification decision. A class probability estimate for data point x is the probability that the true class is c, i.e.,

$$\Pr(f(x) = c|h_l), \text{ for } c = 1, ..., C$$

We can combine the class probabilities of all of the hypotheses so that the class probability of the ensemble is

$$\Pr(f(x) = c) = \frac{1}{L}\Sigma_{l=1}^{L}\Pr(f(x) = c|h_l)$$

The predicted class of x is then the class having the highest class probability.

EXAMPLE 9

Consider the data set in Figure 8.2. Let us examine three classifiers.

Classifier 1 This method finds the centroid of the two classes. The distance from the test pattern P is found from the two centroids. Let this be $d(\mathrm{P}, \mathrm{C}_1)$ and $d(\mathrm{P}, \mathrm{C}_2)$. Then the probability that P belongs to Class 1 will be

$$\Pr(\mathrm{P} \in \text{Class 1}) = 1 - \frac{d(\mathrm{P}, \mathrm{C}_1)}{d(\mathrm{P}, \mathrm{C}_1) + d(\mathrm{P}, \mathrm{C}_2)}$$

Similarly,

$$\Pr(\mathrm{P} \in \text{Class 2}) = 1 - \frac{d(\mathrm{P}, \mathrm{C}_2)}{d(\mathrm{P}, \mathrm{C}_1) + d(\mathrm{P}, \mathrm{C}_2)}$$

If Class 1 is X, $\mathrm{C}_1 = (1.86, 1.4)^t$ and $\mathrm{C}_2 = (5.5, 1.5)^t$.

$$d(\mathrm{P}, \mathrm{C}_1) = 2.22; d(\mathrm{P}, \mathrm{C}_2) = 1.58$$

Then,

$$\Pr(\mathrm{P} \in \text{Class 1}) = 1 - \frac{2.22}{2.22 + 1.58} = 0.416$$

$$\Pr(\mathrm{P} \in \text{Class 2}) = 1 - \frac{1.58}{2.22 + 1.58} = 0.584$$

Classifier 2 If three closest neighbours of P is taken, it will be 3, 6 and 8. Then,

$$\Pr(\mathrm{P} \in \text{Class 1}) = \frac{1}{3} = 0.333$$

$$\Pr(\mathrm{P} \in \text{Class 2}) = \frac{2}{3} = 0.667$$

Combining the two classifiers,

$$\Pr(\mathrm{P} \in \text{Class1}) = \frac{1}{2} \times (0.416 + 0.333) = 0.3745$$

$$\Pr(\mathrm{P} \in \text{Class 2}) = \frac{1}{2} \times (0.584 + 0.667) = 0.6255$$

The pattern P is therefore classified as belonging to Class 2 or Class O.

There are many weighted voting methods. One such technique involves applying least squares regression to find weights that maximise the accuracy of the classifiers on the training data. Weights are generally obtained by measuring the accuracy of each

individual classifier on the training data and constructing weights that are proportional to those accuracies. One technique is to use a likelihood combination in which naive Bayes algorithm is applied to learn the weights for the classifiers. In ADABOOST, as mentioned in Section 8.2.1, the weight for a classifier is computed from its accuracy measured on the weighted training distribution that was used to learn the classifier. A Bayes approach to weighted vote is to compute the posterior probability of each classifier. This method requires the definition of a prior probability $P(h_l)$ which is multiplied with the likelihood $p(S|h_l)$ to estimate the posterior probability of each classifier. The likelihood $p(S|h_l)$ is obtained by applying the hypothesis h_l to the training data set S.

EXAMPLE 10

Let h_1 be the hypothesis that if $x_1 \leq 3$, the pattern belongs to Class X and that the pattern belongs to Class O otherwise. Let the second hypothesis h_2 be that if $x_1 \leq 5$, the pattern belongs to Class X and that it belongs to Class O otherwise. Let the prior probabilities $P(h_1)$ and $P(h_2)$ be 0.5 each. For the hypothesis 1, since 9 patterns are classified correctly in the training data set, the likelihood $P(S|h_1)$ will be 0.9. For the hypothesis 2, since 7 samples are classified correctly in the training data set, the likelihood $P(S|h_2)$ will be 0.7.

The posterior probability for hypothesis 1 will be $\dfrac{0.5 \times 0.9}{0.45 + 0.35} = 0.5625$.

The posterior probability for hypothesis 2 will be $\dfrac{0.5 \times 0.7}{0.45 + 0.35} = 0.4375$.

Another approach to combining classifiers is to learn a gating network or a gating function that takes x as input and produces as output the weights w_l to be applied to compute the weighted vote of the classifiers. One such gating network is of the form

$$z_l = v_l^T x$$

$$w_l = \frac{e^{z_l}}{\Sigma_u e^{z_u}}$$

Here z_l is the dot product of a parameter vector v_l with the input feature vector x. The output weight w_l is the soft-max of the individual z_ls.

Yet another approach to combining classifiers is called *stacking*. Suppose we have L different learning algorithms A_1,..., A_L and a set S of training examples $(x_1, \theta_1), ..., (x_N, \theta_N)$. Applying each of the algorithms on the training examples, we get hypothesis $h_1, ..., h_L$. The goal of stacking is to learn a good combining classifier h^* such that the final classification will be computed by $h^*(h_1(x), ..., h_L(x))$. The following scheme can be used to learn h^*.

Let $h_l^{(-i)}$ be a classifier constructed by algorithm A_l applied to all the training

examples in S except example i. This means each algorithm is applied to the training data m times, leaving out one training example each time. Then each classifier $h_l^{(-i)}$ is applied to example x_i to obtain the predicted class θ_i^l. This gives a new data set containing ''level 2'' examples whose features are the classes predicted by each of the L classifiers. Each example has the form

$$< (\theta_i^1, \theta_i^2, ..., \theta_i^L), \theta_i >.$$

Now some other learning algorithm can be applied to this level 2 data to learn h^*.

EXAMPLE 11

Table 8.2 Level 2 patterns

Hypothesis	h_1	h_2	h_3	h_4	Label
Pattern 1	X	X	X	X	X
Pattern 2	X	X	X	X	X
Pattern 3	X	X	X	X	X
Pattern 4	X	X	X	X	X
Pattern 5	X	X	X	X	X
Pattern 6	O	O	O	O	O
Pattern 7	O	O	O	O	O
Pattern 8	O	O	O	O	O
Pattern 9	O	O	O	O	O
Pattern 10	O	O	O	O	O

Consider Figure 8.2. Using this data, let us consider different hypothesis for classification. Let h_1 be according to the nearest neighbour. Let h_2 be according to the majority vote for the closest three neighbours. Let h_3 be the classification according to the closest neighbour in the f_1 direction. Let h_4 be the classification according to the nearest centroid. Table 8.2 gives the classification for each pattern when it is left out and the different hypothesis used to classify the pattern.

If we take point P as the test pattern, its level 2 pattern will be (O O X O). If we want to classify this pattern against the level 2 patterns in the training set, using the nearest neighbour algorithm, it will be classified as belonging to class O.

Discussion

This chapter discusses how to combine the decision taken by a number of classifiers. Each classifier is designed using a different sub-set of the data. There are a number of ways of choosing the data set used for each classifier. Once the different classifiers have made a classification for a new pattern, these classifications have to be combined to give the final classification of the pattern. There are a number of ways of doing this as detailed in this chapter.

Further Reading

Combination of classifiers is discussed by Dietterich (1997), Chen and Karnel (2009) and Ho et al. (1994). The technique of boosting is explained by Drucker et al. (1994). The ADABOOST algorithm has been described by Freund and Schapire (1997; 1999). Other methods for combination of classifiers are reviewed by Santos et al. (2008), Huang and Suen (1995), Chin and Kamel (2009), Saranli and Demirekler (2001), Wang and Wang (2006), Windeatt (2003), Xu et al. (1992) and Zouari et al. (2005).

Exercises

1. Consider the collection of two-dimensional patterns :

   $X_1 = (1, 1, 1), X_2 = (1, 2, 1), X_3 = (2, 1, 1), X_4 = (2, 1.5, 1),$
   $X_5 = (3, 2, 1), X_6 = (4, 1.5, 2), X_7 = (4, 2, 2), X_8 = (5, 1.5, 2),$
   $X_9 = (4.5, 2, 2), X_{10} = (4, 4, 3), X_{11} = (4.5, 4, 3), X_{12} = (4.5, 5, 3),$
   $X_{13} = (4, 5, 3), X_{14} = (5, 5, 3)$

   where each pattern is represented by feature 1, feature 2 and the class label. Consider a test pattern at (3.5, 2.8). Implement bagging on the data sets and classify the test pattern. Consider two patterns from each class at random and use the nearest neighbour algorithm to classify the test pattern. Do this procedure 5 times and classify the test pattern according to majority vote.

2. Consider the data set in Exercise 1. The following sub-sets have been taken from the data set:

   $S_1 = \{1, 3\}, S_2 = \{2, 4, 5\}, S_3 = \{6, 7\},$
   $S_4 = \{8, 9\}, S_5 = \{10, 11\}, S_6 = \{12, 13, 14\}$

   Consider a test point at (3.5, 2.8). Leave out one sub-set from each class and classify the test pattern. Consider all possible data sets by leaving out one sub-set from each class and classify the test pattern according to majority vote.

3. Consider the data set in Exercise 1. Classify the test pattern by leaving out feature 1. Similarly, classify the test pattern by leaving out feature 2.

4. Consider the data set in Exercise 1. Classify a test pattern at (3.5, 3) by the method of stacking using the following hypotheses:

   (a) Classify according to nearest neighbour.

(b) Classify according to majority class of three neighbours.

(c) Take the distance from the centroid of the classes and classify according to the class of the closest centroid.

5. Consider the collection of two-dimensional patterns:

$$X_1 = (1, 1, 1), X_2 = (1, 2, 1), X_3 = (2, 1, 1), X_4 = (2, 1.5, 1),$$
$$X_5 = (3, 2, 1), X_6 = (3.5, 2, 2), X_7 = (4, 1.5, 2), X_8 = (4, 2, 2),$$
$$X_9 = (5, 1.5, 2), X_{10} = (4.5, 2, 2)$$

where each pattern is represented by feature 1, feature 2 and the class label. Classify a pattern at (3.5, 1) by using the ADABOOST algorithm with the following weak classifiers:

(a) If $(f_1 < 3)$ then the pattern belongs to Class 1
else the pattern belongs to Class 2

(b) If $(f_1 \leq 3.5)$ then the pattern belongs to Class1
else the pattern belongs to Class 2

(c) If $((4f_1 + 5f_2) \leq 20)$ then the pattern belongs to Class 1
else the pattern belongs to Class 2

Computer Exercise

1. Write a program to carry out bagging on a data set. Use the program to choose two patterns from each class for the data set in Figure 8.3. Do this a number of times and run the nearest neighbour classification program on the data sets. Write a program for combination of classifiers and use the program to find the final classification of the test patterns.

2. Implement the ADABOOST program and use it on the data set given in Figure 8.3. Use some decision stumps or weak classifiers. Use four different classifiers and classify some test patterns using this program.

3. Implement the algorithm for error correcting output coding. Use the data set in Figure 8.3. Randomly partition the data set into two sub-sets a number of times and use the combination to classify test patterns.

4. Take the data set in Figure 8.3. Obtain a decision tree for the data set. Using the Markov chain Monte Carlo method, obtain different hypotheses by altering the

decision tree. Run it for a number of hypotheses and obtain a set of 3 classifiers. Use these classifiers on the data set to classify test patterns using majority voting.

5. Take a large data set with training data having more than 5000 patterns. Use different methods of classification. For each classifier, note all the performance measures. Compare the different classifiers and choose the best classifier for the data set being used by you.

Bibliography

1. Bryll, Robert, Ricardo Gutierrez-Osuna and Francis Quek. Attribute bagging: Improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition* 36(6): 1291–1302. 2003.
2. Chen, Lei and Mohamed S. Karnel. A generalized adaptive ensemble generation and aggregation approach for multiple classifier systems. *Pattern Recognition* 42(5): 629–644. 2009.
3. Dietterich, T.G. Machine learning research: Four current directions. *AI Magazine* 18(4): 97–136. 1997.
4. Drucker, H., C. Cortes, L. D. Jackel, Y. Lecun, V. Vapnik. Boosting and other ensemble methods. *Neural Computation* 6(6): 1289–1301. 1994.
5. Freund, Yoav and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* 55(1): 119–139. 1997.
6. Freund, Yoav and Robert E. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence* 14(5): 771–780. 1999.
7. Ho, Tin Kam, Jonathan J. Hull and Sargur N. Srihari. Decision combination in multiple classifier systems. *IEEE Trans on PAMI* 16(1). 1994.
8. Hothorn, Torsten and Berthold Lausen. Double-bagging: Combining classifiers by bootstrap aggregation. *Pattern Recognition* 36(6): 1303–1309. 2003.
9. Huang, Y. S. and C.Y. Suen. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 17: 90–94. 1995.
10. Santos, Eulanda M. Dos, Robert Sabourin and Patrick Maupin. A dynamic overproduce-and-choose strategy for the selection of classifier ensembles. *Pattern Recognition* 41(10): 2993–3009. 2008.

11. Saranli, Afsar and Mbeccel Demirekler. A statistical unified framework for rank-based multiple classifier decision combination. *Pattern Recognition* 34(4): 865–884. 2001.

12. Wang, Xiao and Han Wang. Classification by evolutionary ensembles. *Pattern Recognition* 39(4): 595–607. 2006.

13. Windeatt, Terry. Vote counting measures for ensemble classifiers. *Pattern Recognition* 36(12): 2743–2756. 2003.

14. Xu, Lei, Adam Krzyzak and Ching Y. Suen. Methods of combining multiple classifiers and their application to handwriting recognition. *IEEE Trans on SMC* 22(3). 1992.

15. Zouari, Hla, Laurent Heutte and Yves Lecourtier. Controlling the diversity in classifier ensembles through a measure of agreement. *Pattern Recognition* 38(11): 2195–2199. 2005.

9 Clustering

Learning Objectives

The objective of this chapter is to provide an introduction to various clustering paradigms and associated algorithms. After reading the chapter, you will understand the following.

- What is clustering?
- Why is clustering necessary?
- The stages in clustering
- Hierarchical clustering which generates a hierarchy of partitions including
  - Divisive clustering
  - Agglomerative clustering
- Partitional clustering which obtains a single partition of the data set
  - k-means algorithm
- Soft partitioning where a pattern can be a member of more than one cluster
- Stochastic search techniques for clustering
- Clustering large data sets
  - incremental clustering
  - use of intermediate abstraction
  - divide-and-conquer algorithm

In this chapter, we will deal with the process of clustering, providing a special emphasis on various frequently used clustering algorithms.

Clustering is the process of grouping a set of patterns. It generates a partition consisting of cohesive groups or clusters from a given collection of patterns as depicted in Figure 9.1. Representations or descriptions of the clusters formed are used in decision making—classification is one of the popular decision-making paradigms used.

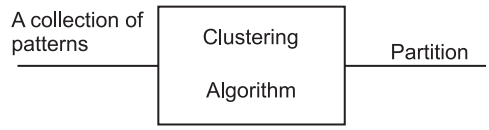


Figure 9.1 The input–output behaviour of a clustering algorithm

Patterns to be clustered are either labelled or unlabelled. Based on this, we have:

- *Clustering algorithms which typically group sets of unlabelled patterns* These types of paradigms are so popular that clustering is viewed as an *unsupervised classification of unlabelled patterns.*
- *Algorithms which cluster labelled patterns* These types of paradigms are practically important and are called *supervised clustering* or *labelled clustering.* Supervised clustering is helpful in identifying clusters within collections of labelled patterns. Abstractions can be obtained in the form of cluster representatives/descriptions which are useful for efficient classification.

The process of clustering is carried out so that patterns in the same cluster are *similar* in some sense and patterns in different clusters are *dissimilar* in a corresponding sense. This may be illustrated with the help of a two-dimensional data set shown in Figure 9.2.

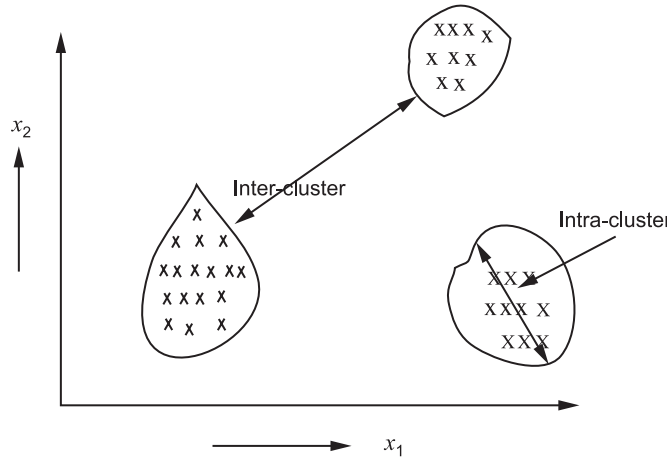


Figure 9.2 The input–output behaviour of clustering

Here, each pattern is represented as a point in two-dimensional space. There are three clusters as can be ascertained visually. Further, the Euclidean distance between any two points belonging to the same cluster is smaller than that between any two points belonging to different clusters. This notion characterises similarity—*intra-*

cluster distance (similarity) is small (high) and *inter-cluster distance (similarity) is large (low)*. We illustrate this idea with the following example.

EXAMPLE 1

Let us consider a two-dimensional data set of 10 vectors given by

$$\mathrm{X}_1 = (1,1) \quad \mathrm{X}_2 = (2,1) \quad \mathrm{X}_3 = (1,2)$$
$$\mathrm{X}_4 = (2,2) \quad \mathrm{X}_5 = (6,1) \quad \mathrm{X}_6 = (7,1)$$
$$\mathrm{X}_7 = (6,2) \quad \mathrm{X}_8 = (6,7) \quad \mathrm{X}_9 = (7,7)$$
$$\mathrm{X}_{10} = (7,6)$$

as shown in Figure 9.3.

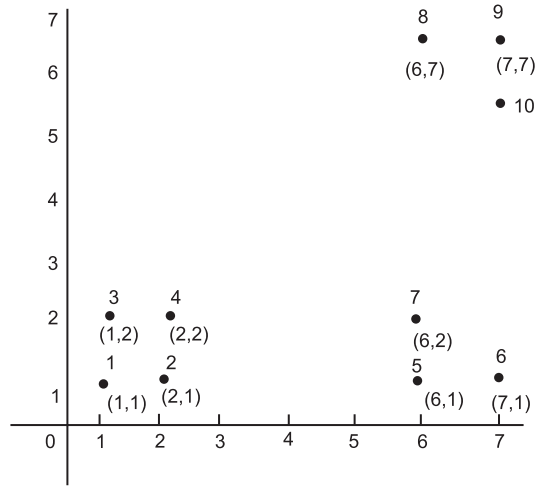


Figure 9.3 A two-dimensional data set of 10 vectors

Let us say that any two points belong to the same cluster if the distance between them is less than a threshold. Specifically, in this example, we use the squared Euclidean distance to characterise the distance between the points and use a threshold of 5 units to cluster them. The squared Euclidean distance between two points x_i and x_j is defined as

$$d(x_i, x_j) = \sum_{l=1}^{d} (x_{il} - x_{jl})^2, \text{ where } d \text{ is the dimensionality of the points.}$$

It is possible to represent the squared Euclidean distance between pairs of points using the matrix shown in Table 9.1, called the distance matrix. Note that the value of d is 2 here because we are considering 2-dimensional patterns.

Table 9.1 The ijth entry in the matrix is the distance between X_i and X_j

	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}
X_1	0	1	1	2	25	36	26	61	72	61
X_2	1	0	2	1	16	25	17	52	61	50
X_3	1	2	0	1	26	37	25	50	61	52
X_4	2	1	1	0	17	26	16	41	50	41
X_5	25	16	26	17	0	1	1	36	37	26
X_6	36	25	37	26	1	0	2	37	36	25
X_7	26	17	25	16	1	2	0	25	26	17
X_8	61	52	50	41	36	37	25	0	1	2
X_9	72	61	61	50	37	36	26	1	0	1
X_{10}	61	50	52	41	26	25	17	2	1	0

Note that, in this case, the clusters are clearly reflected in the matrix itself. There are 3 sub-matrices of sizes 4×4, 3×3, and 3×3, as can be seen in the table which satisfy the condition that the value of any entry in the sub-matrix is less than 5 units. For example, the sub-matrix of size 4×4, which corresponds to the first four patterns, has entries in the range 0 to 2—every other entry in the first four rows is larger than 5. In a similar manner, it can be inferred that patterns X_5, X_6, and X_7 can be grouped into one cluster and the remaining three patterns, that is patterns X_8, X_9, and X_{10}, into the third cluster. So, the three clusters are

Cluster 1: $\{X_1, X_2, X_3, X_4\}$

Cluster 2: $\{X_5, X_6, X_7\}$

Cluster 3: $\{X_8, X_9, X_{10}\}$

Note the following points:

1. In general, the clustering structure may not be so obvious from the distance matrix. For example, by considering the patterns in the order X_1, X_5, X_2, X_8, X_3, X_6, X_9, X_4, X_7, X_{10} (refer to Exercise 1), the sub-matrices corresponding to the three clusters are not easy to visualise.
2. In this example, we have used the squared Euclidean distance to denote the distance between two points. It is possible to use other distance functions also.
3. Here, we defined each cluster to have patterns such that the distance between any two patterns in a cluster (intra-cluster distance) is less than 5 units and the distance between two points belonging to two different clusters (inter-cluster distance) is greater than 5 units. This is a very simple way of characterising clusters; in general, we may have different notions of a cluster. It gets instantiated by the algorithm used for clustering.

Clustering is a subjective activity. It is possible that different users require different partitions of the same data set. For example, by adding some more points to the data shown in Figure 9.2, it is possible to change the intra-cluster distance in one of the clusters to that shown in Figure 9.4. As a consequence, the notion of similarity depicted in Figure 9.2 does not hold with respect to the clusters shown in Figure 9.4.

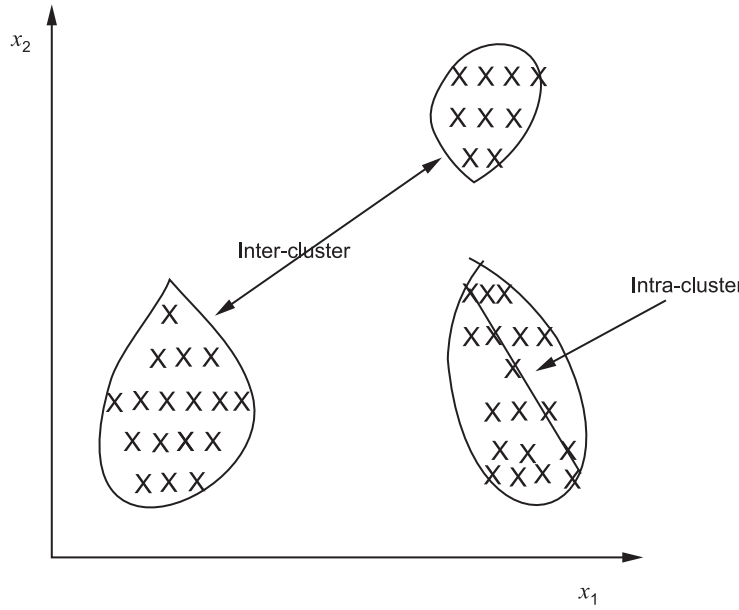


Figure 9.4 A different notion of similarity

Here also we have three clusters; however, points in each cluster satisfy the property that a point is in the same cluster as its nearest neighbour.

Partition generated by clustering is either hard or soft. A *hard clustering algorithm* generates clusters that are non-overlapping. For example, the clusters shown in Figure 9.2 and Figure 9.4 form a non-overlapping or hard partition. It is possible to generate a *soft partition* where a pattern belongs to more than one cluster. In such a case, we will have overlapping clusters.

Consider a set of patterns $\mathcal{X} = \{x_1, x_2, \cdots, x_n\}$, where the ith cluster $X_i \subset \mathcal{X}$, $i = 1, \cdots, C$, is such that $\cup_{i=1}^{C} X_i = \mathcal{X}$, and no $X_i = \phi$. If in addition, $X_i \cap X_j = \phi$, for all i and j, $i \neq j$, then we have a hard partition. The number of hard partitions itself could be prohibitively large. For example, for a collection of 3 patterns A, B, and C, we have the following 2-partitions: {{A}, {B,C}}; {{B}, {A,C}}; {{C}, {A,B}}; note that $n = 3$ here and therefore, there are 3 2-partitions. This may be worked out as follows. Consider all the sub-sets of the set {A, B, C}—there are 8 of them.

ϕ: (empty set)

{A}, {B}, {C}: (one-element sub-sets)

{A, B}, {B, C}, {A, C}: (two-element subsets)

{A, B, C}: (improper sub-set)

Each 2-partition has two clusters; further, a cluster is non-empty and is a proper sub-set of the set of patterns. Note that there are 6 $(2^3 - 2)$ non-empty proper sub-sets; because out of the 8 sub-sets, one is empty (ϕ) and one ({A, B, C}) is not a proper sub-set. So, each of these 6 sub-sets can be a cluster and the other cluster in the 2-partition is the complement of the cluster. A detailed listing of the possible 2-partitions is given in Table 9.2.

Table 9.2 A listing of 2-partitions of the set {A,B,C}

Serial number	Non-empty sub-set	Its complement	Corresponding 2-partition
1	{A}	{B, C}	{{A}, {B, C}
2	{B}	{A, C}	{{B}, {A, C}
3	{C}	{A, B}	{{C}, {A, B}
4	{A, B}	{C}	{{A, B}, {C}
5	{B, C}	{A}	{{B, C}, {A}
6	{A, C}	{B}	{{A, C}, {B}

Note that each of the 2-partitions is repeated. For example, the set of two clusters ({A}, {B, C}) corresponding to Row 1 is repeated in Row 5. Similarly, observe the repetitions of 2-partitions in Rows 2 and 6 ({B}, {A, C}), and in Rows 3 and 4 ({C}, {A, B}). By considering each of the 2-partitions only once, we have only 3 (half of the 6 partitions; $\frac{2^3-2}{2} = 2^2 - 1$) distinct 2-partitions. In general, for a set of n patterns, it is possible to show that the number of 2-partitions is $2^{n-1} - 1$ (refer to Exercise 2).

It is interesting to consider partitions of more than 2 clusters. This is denoted as the number of partitions of n patterns into m blocks (clusters) and is given by

$$\frac{1}{m!}\sum_{i=1}^{m}(-1)^{m-i}\binom{m}{i}(i)^n$$

Consider a very small size problem of grouping 15 patterns into 3 clusters. The possible number of partitions is 2, 375, 101. So, exhaustive enumeration of all possible partitions to decide the *best* partition, in some sense, is not practical while dealing with large collections of patterns.

Knowledge is implicit in clustering. So, partitioning is done based on the domain knowledge or user-provided information in different forms; this could include choice of similarity measure, number of clusters, and threshold values. Further, because of

the well-known *theorem of the ugly duckling*, clustering is not possible without using extra-logical information.

The theorem of the ugly duckling states that the number of predicates shared by any two patterns is constant when all possible predicates are used to represent patterns. So, if we judge similarity based on the number of predicates shared, then any two patterns are equally similar. This may be illustrated with the data shown in Table 9.3.

Table 9.3 Example data set

Pattern/feature	f1	f2
P1	0	0
P2	0	1
P3	1	0
P4	1	1

There are four object types, P1, P2, P3, and P4, based on two features f1 and f2; if the first three of them, that is, P1, P2, and P3, are considered, then there are at most 8 predicates. This may be illustrated as shown in Table 9.4. By considering all the four object types, there will be at most 16 predicates (refer to Exercise 3).

Table 9.4 Example data set with eight features (predicates)

Pattern/feature	f1	f2	f1∧¬f2	¬f2	¬f1∧f2 (g2)	¬f1 (g1)	f1v¬f2	¬f1vf2
P1	0	0	0	1	0	1	1	1
P2	0	1	0	0	1	1	0	1
P3	1	1	0	0	0	0	1	1

The logical connectives used are given in Table 9.5. Note that with every object type, there are at most two predicates: f1 and ¬f1; with two object types, there are at most four predicates: f1, f2, ¬f1, and ¬f2.

Table 9.5 Logical connectives

Symbol	Logical connective
∧	logical and
∨	inclusive or
¬	negation
⊕	exclusive or

Note that any pair of object types match on 4 out of the 8 predicates (extracted features). One may argue that f1 and f2 are primitive or original features and so more importance should be given to them. But in the absence of any extra information, it is

difficult to choose a sub-set of the possible features. For example, if one considers g1 =(¬f1) and g2 = ⊕ (f1,f2) as primitive features, then f1 = ¬g1 and f2 = ⊕ (¬g1,g2). So, for a computer, g1 and g2 are as primitive as f1 and f2. As a consequence, one needs to use extra-logical evidence to give more importance to some features over the others. This means, in the strictest sense *unsupervised classification* is not possible. So, different clustering algorithms use knowledge in different forms to generate an *intended* partition (partition based on the user's interest) of the data.

Clustering is useful for generating data abstraction. The process of data abstraction may be explained using Figure 9.5. As explained in Chapter 2, a cluster of points is represented by its *centroid* or its *medoid*. The centroid stands for the sample mean of the points in cluster C; it is given by $\frac{1}{N_C}\sum X_i \in C$, where N_C is the number of patterns in cluster C and it need not coincide with one of the points as indicated in the figure. The medoid is the most centrally located point in the cluster; more formally, the medoid is that point in the cluster from which the sum of the distances from the points in the cluster is the minimum. There is another point in the figure which is far off from any of the points in the cluster. This is an *outlier*. The centroid of the data can shift without any bound based on the location of the outlier; however, the medoid does not shift beyond the boundary of the original cluster irrespective of the location of the outlier. So, clustering algorithms that use medoids are more *robust* in the presence of noisy patterns or outliers. This may be illustrated with the help of the following example.

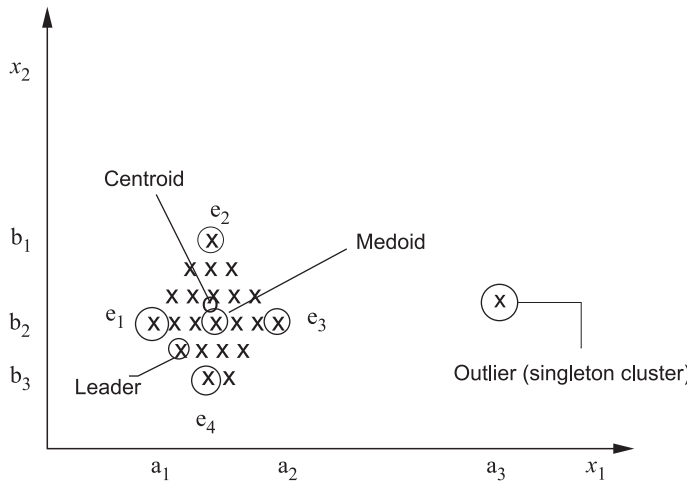


Figure 9.5 Representing clusters using centroids and medoids

EXAMPLE 2

Consider the cluster of five patterns given below.

$X_1 = (1, 1)$ $X_2 = (1, 2)$ $X_3 = (2, 1)$
$X_4 = (1.6, 1.4)$ $X_5 = (2, 2)$

The centroid of the cluster is the sample mean

$$\mu = \tfrac{1}{5}[\ (1,\ 1) + (1,\ 2) + (2,\ 1) + (1.6,\ 1.4) + (2,\ 2)]$$

So, $\mu = (1.52,\ 1.48)$; note that μ is not one of the X_is ($i = 1,\ldots,5$).

However, the medoid of the cluster is $m = (1.6,\ 1.4)$; note that the medoid of a cluster of points is, by definition, a centrally located pattern of the cluster such that the sum of the distances from points in the cluster is minimum. In our example, the distance of the five points from m, using squared Euclidean distance for distance computation, is

$$d(X_1, m) = 0.52$$
$$d(X_2, m) = 0.72$$
$$d(X_3, m) = 0.32$$
$$d(X_4, m) = 0.0$$
$$d(X_5, m) = 0.52$$

The sum of the distances = 2.08.

Consider the possibility of any other pattern being the medoid. For example, consider X_1. The corresponding distances of points in the cluster to X_1 are

$$d(X_1, X_1) = 0.0$$
$$d(X_2, X_1) = 1.0$$
$$d(X_3, X_1) = 1.0$$
$$d(X_4, X_1) = 0.52$$
$$d(X_5, X_1) = 2.0$$

The sum of the distances = 4.52.

So, by considering the possibility of having X_1 as the medoid, we get the sum of the distances to points in the cluster as 4.52. This is larger than 2.08 which was obtained using X_4 as the possible medoid. Similarly, it is possible to show that the sum of the

distances of points in the cluster from X_2, X_3, or X_5 will be larger than 2.08 (refer to Exercise 4), ruling out the possibility of any point other than X_4 being the medoid.

Now consider addition of a point (1.6, 7.4) to the cluster. The centroid of the updated cluster (with 6 points) is

$$\mu^1 = \frac{1}{6}\,[(1,1) + (1,2) + (2,1) + (1.6,1.4) + (2,2) + (1.6,7.4)]$$

So, μ^1 = (1.53, 2.47); centroid shifts from μ to μ^1. It is possible to show that the medoid shifts from m = (1.6,1.4) to m^1 = (2, 2) (refer to Exercise 5).

Now consider the addition of a point (1.6, 17.4) to the original cluster, instead of (1.6, 7.4). The centroid of the resulting cluster of 6 points is

$$\mu^2 = \frac{1}{6}[(1,1) + (1,2) + (2,1) + (1.6,1.4) + (2,2) + (1.6,17.4)]$$

So, μ^2 = (1.53, 4.13). The centroid keeps shifting away based on the location of the outlier pattern which is added to a cluster. Note the variation of μ to μ^1 and to μ^2 based on the outlier pattern in the cluster. However, the medoid m^2 of the cluster after adding(1.6,17.4) is (2, 2). So, the medoid of a cluster is more robust to the addition of an outlier as m^2 is the same as m^1.

It is possible to have more than one representative per cluster; for example, four extreme points labelled e_1, e_2, e_3, e_4 can represent the cluster as shown in Figure 9.5. It is also possible to use a logical description to characterise the cluster. For example, using a conjunction of disjunctions, we have the following description of the cluster:

$$(x = a_1, ..., a_2) \wedge (y = b_1, ..., b_3)$$

9.1 Why is Clustering Important?

Clusters and their generated descriptions are useful in several decision-making situations like classification, prediction, etc. The number of cluster representatives obtained is smaller than the number of input patterns and so there is data reduction. This may be illustrated using the following example.

EXAMPLE 3

Consider the two-dimensional data set of 16 labelled patterns given below. Let the two classes be labelled ‘‘X’’ and ‘‘O’’.

X : (1, 1), (1, 2), (2,1), (2, 2), (1, 5), (1, 6), (2, 5), (2, 6)

O : (6, 1), (6, 2), (7, 1), (7, 2), (6, 6), (6, 7), (7, 6), (7, 7)

Let (2, 3) be a test pattern which needs to be classified using the NN on the above 16 labelled patterns. Note that (2, 2) is the nearest neighbour of (2, 3) with a squared Euclidean distance of 1 unit between them. This can be obtained by computing 16 distance values—calculating the squared Euclidean distance between the test pattern (2, 3) and each of the 16 labelled patterns. So, (2, 3) is classified as belonging to class X because its nearest neighbour (2, 2) belongs to X.

However, by clustering the 16 patterns using a suitable clustering algorithm and representing each resulting cluster by its centroid, we can reduce the number of distance values to be computed, from the number of labelled patterns to the number of cluster representatives. This may be illustrated as follows.

Let the 16 patterns be clustered using the same criterion as the one used in Example 1, that is, the squared Euclidean distance between any two patterns placed in the same cluster should be within a threshold of 5 units. This results in 4 clusters; two from each of the two classes. The two clusters corresponding to class ''X'' are:

C_1 = { (1, 1), (1, 2), (2, 1), (2, 2)} and

C_2 = { (1, 5), (1, 6), (2, 5), (2, 6)}.

Similarly, the two clusters corresponding to class ''O'' are:

C_3 = { (6, 1), (6, 2), (7, 1), (7, 2)} and

C_4 = { (6, 6), (6, 7), (7, 6), (7, 7)}.

Let each of the clusters be represented by their centroids; the centroids are given by

Centroid of C_1 = (1.5, 1.5) and

Centroid of C_2 = (1.5, 5.5)

corresponding to class ''X'' and the centroids of the two clusters of patterns in class ''O'' are:

Centroid of C_3 = (6.5, 1.5) and

Centroid of C_4 = (6.5, 6.5).

So, instead of using the 16 labelled patterns, if one uses the centroids of the 4 clusters as the labelled data, then there is a reduction in the data to be analysed by a factor of 4. The reduced labelled data is

X: (1.5, 1.5), (1.5, 5.5)

O: (6.5, 1.5), (6.5, 6.5).

Using this data, the nearest neighbour of (2, 3) is (1.5, 1.5) from class X; so the test pattern is classified as belonging to X using only 4 distance computations, the distance between the test pattern and each of the 4 centroids. Also, note that for computing the distances from the test pattern, we need to store only the centroids of the 4 clusters. So, there is a reduction in both space and time. Observe that in our illustration, there is a 75% reduction in both the time and space—we need to compute only 4 distances instead of 16 and store only 4 centroids instead of 16 patterns.

One may argue that clustering the labelled data and computing the centroids can be time and space consuming. However, clustering can be done beforehand, that is, as soon as the labelled data is made available. Further, the process of clustering and computation of cluster centroid is done only once. Once we have the representatives and are given a test pattern to be classified, we can use the cluster representatives as labelled data to reduce both the time and space requirements significantly. Such a clustering-based classification scheme is very useful in solving large-scale pattern classification problems encountered in data mining.

Clustering can also be used on the set of features in order to achieve dimensionality reduction which is useful in improving the classification accuracy at a reduced computational resource requirement. This strategy is very useful in classification using NN. Clustering is associated with grouping unlabelled patterns. However, there are several applications where it makes sense to cluster labelled patterns. For example, the branch-and-bound scheme for finding out the nearest neighbour efficiently (described in Chapter 3) exploits clustering.

EXAMPLE 4

Consider two classes of patterns labelled as ''0'' and ''3''. Let some samples be as shown in Figure 9.6. There are two ''3''s and three ''0''s. If we use a clustering algorithm to generate 4 clusters out of the given 5 patterns, it may select a cluster corresponding to each of the ''0''s and another cluster corresponding to the ''3''s. If we are interested in having equal representation from each class, then it is more meaningful to cluster patterns in each class separately to produce two clusters; in such a case, we will have 2 ''0'' clusters and 2 ''3'' clusters.

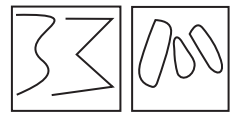


Figure 9.6 Labelled clustering

Clustering labelled patterns is useful in *prototype selection*. It is possible to group patterns belonging to each class separately. The representatives of the resulting

clusters can then form prototypes of the class. By using these prototypes rather than the original data for classification, we can reduce the computational requirements.

One of the important applications of clustering is in *data re-organisation*.

EXAMPLE 5

Data re-organisation may be illustrated using the binary pattern matrix shown in Table 9.6. The input data set contains four four-dimensional binary patterns. The patterns are labelled 1 to 4 and the features are f1 to f4. By clustering the patterns using rows as the criteria, we can get the data set shown in Table 9.7 where similar patterns are put together.

Table 9.6 Input data

Pattern	f1	f2	f3	f4
1	1	0	0	1
2	0	1	1	0
3	0	1	1	0
4	1	0	0	1

Table 9.7 Row-clustered data

Pattern	f1	f2	f3	f4
1	1	0	0	1
4	1	0	0	1
2	0	1	1	0
3	0	1	1	0

Table 9.8 Clustering based on rows and columns

Pattern	f1	f4	f2	f3
1	1	1	0	0
4	1	1	0	0
2	0	0	1	1
3	0	0	1	1

Now by clustering the data set in Table 9.7 by columns and placing similar columns together, we get the data set shown in Table 9.8. Note that this table clearly reflects

the structure in the data. The cluster consisting of patterns 1 and 4 can be described using the conjunction $f1 \wedge f4$. Similarly, the second cluster having patterns 2 and 3 can be described by $f2 \wedge f3$.

Another important application of clustering is in identifying outliers. For example, by clustering the data shown in Figure 9.5, the outlier will show up as an element of a singleton cluster. It is possible to realise this using any of the popular clustering algorithms. This forms the basis for a variety of applications like automatic auditing of records in a database and detection of intrusions. It is also possible to use clustering for guessing missing values in a pattern matrix.

For example, let the value of the jth feature of the ith pattern be missing. We can still cluster the patterns based on features other than the jth one. Based on this grouping, we find the cluster to which the ith pattern belongs. The missing value can then be estimated based on the jth feature value of all the other patterns belonging to the cluster.

One of the major difficulties encountered in clustering is the problem of cluster tendency, wherein the input data are examined to see if there is any merit to performing a cluster analysis prior to performing the actual clustering. It is strongly felt that data which do not contain clusters should not be processed by a clustering algorithm. However, some kind of partitioning may be useful even when there are no ''clusters''. For example, the quick sort algorithm partitions data sets in each pass using the divide-and-conquer strategy to reduce computational complexity; it can exhibit a good average performance even when the data to be sorted is uniformly distributed. So, in applications like data mining where one of the major goals is to realise a partition that helps in building an efficient indexing scheme, clustering can play a useful role even when ''clusters'' are not apparent in the data. This is because clustering can still facilitate decision making at a reduced computational effort.

Clustering has found applications in a large number of areas. These include biometrics, document analysis and recognition, information retrieval, bioinformatics, remote-sensed data analysis, biomedical data analysis, target recognition, and data mining. Some of the approaches for clustering of large data sets will be discussed in the last section.

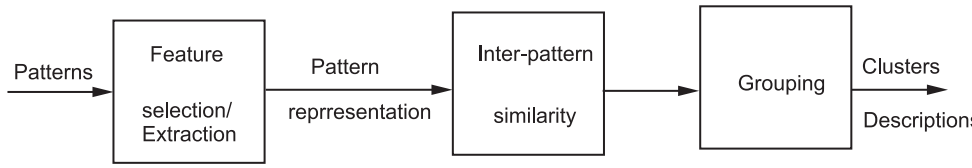


Figure 9.7 Stages in clustering

The important steps in the clustering process as depicted in Figure 9.7 are

1. Pattern representation

2. Definition of an appropriate similarity/dissimilarity function
3. Selecting a clustering algorithm and using it to generate a partition/description of the clusters
4. Using these abstractions in decision making

The first two components were discussed in Chapter 2. In the rest of the chapter, we will describe several clustering algorithms with an emphasis on their capacity to handle large volumes of data. There is a wide variety of clustering algorithms. Different approaches can be described with the help of the hierarchy shown in Figure 9.8.

At the top, there is a distinction between hard and soft clustering paradigms based on whether the partitions generated are overlapping or not. Hard clustering algorithms are either hierarchical, where a nested sequence of partitions is generated, or partitional where a partition of the given data set is generated. Soft clustering algorithms are based on fuzzy sets, rough sets, artificial neural nets (ANNs),or evolutionary algorithms, specifically genetic algorithms (GAs).

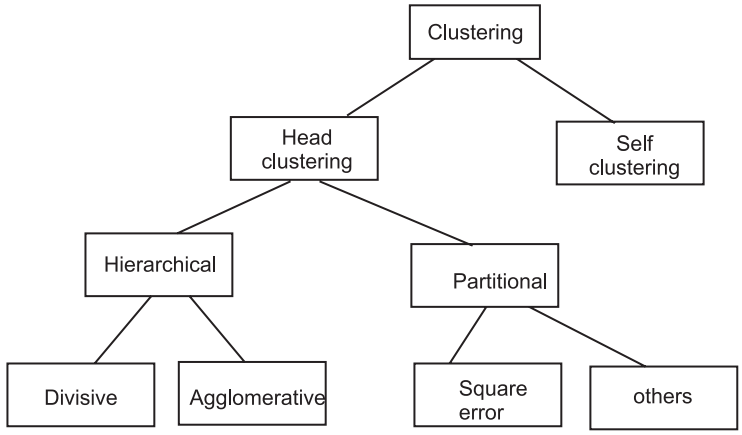


Figure 9.8 Taxonomy of clustering approaches

9.2 Hierarchical Algorithms

Hierarchical algorithms produce a nested sequence of data partitions. The sequence can be depicted using a tree structure that is popularly known as a *dendrogram*. The algorithms are either *divisive* or *agglomerative*. The former starts with a single cluster having all the patterns; at each successive step, a cluster is split. This process continues till we end up with one pattern in a cluster or a collection of singleton clusters.

Divisive algorithms use a *top-down* strategy for generating partitions of the data. Agglomerative algorithms, on the other hand, use a *bottom-up* strategy. They start with n singleton clusters when the input data set is of size n, where each input pattern is in a different cluster. At successive levels, the most similar pair of clusters is merged to reduce the size of the partition by 1. An important property of agglomerative algorithms is that once two patterns are placed in the same cluster at a level, they remain in the same cluster at all subsequent levels. Similarly, in the divisive algorithms, once two patterns are placed in two different clusters at a level, they remain in different clusters at all subsequent levels.

9.2.1 Divisive Clustering

Divisive algorithms are either *polythetic* where the division is based on more than one feature or *monothetic* when only one feature is considered at a time. A scheme for polythetic clustering involves finding all possible 2-partitions of the data and choosing the best partition. Here, the partition with the least sum of the sample variances of the two clusters is chosen as the best. From the resulting partition, the cluster with the maximum sample variance is selected and is split into an optimal 2-partition. This process is repeated till we get singleton clusters.

The sum of the sample variances is calculated in the following way. If the patterns are split into two partitions with m patterns $X_1,\ldots, X_m$ in one cluster and n patterns $Y_1,\ldots,Y_n$ in the other cluster with the centroids of the two clusters being C1 and C2 respectively, then the sum of the sample variances will be

$$\sum_i (X_i - C_1)^2 + \sum_j (Y_j - C_2)^2$$

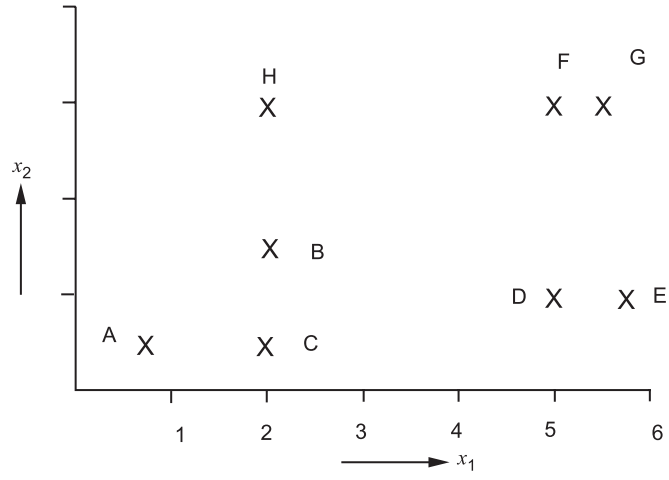


Figure 9.9 Eight 2-dimensional points

EXAMPLE 6

Figure 9.9 shows 8 two-dimensional patterns. The patterns are as follows :

A = (0.5, 0.5); B = (2, 1.5); C = (2, 0.5); D = (5, 1);

E = (5.75, 1); F = (5, 3); G = (5.5, 3); H = (2, 3).

The corresponding dendrogram using divisive clustering is shown in Figure 9.10.

Note that at the top there is a single cluster consisting of all the eight patterns. By considering all possible 2-partitions ($2^7 - 1 = 127$ partitions), the best 2-partition given by {{A, B, C, H}, {D, E, F, G}} is obtained which is shown in the dendrogram. At the next level, the cluster {D, E, F, G} is selected to split into two clusters {D, E} and {F, G}. Note that the cluster {A, B, C, H} cannot be split into two clusters so conveniently; it may be easier to generate a 3-partition of {A, B, C, H} than a two-partition (refer to Exercise 6).

After splitting{D, E, F, G}, the three clusters at the corresponding level of the dendrogram are:

{A, B, C, H}, {D, E}, and {F, G}.

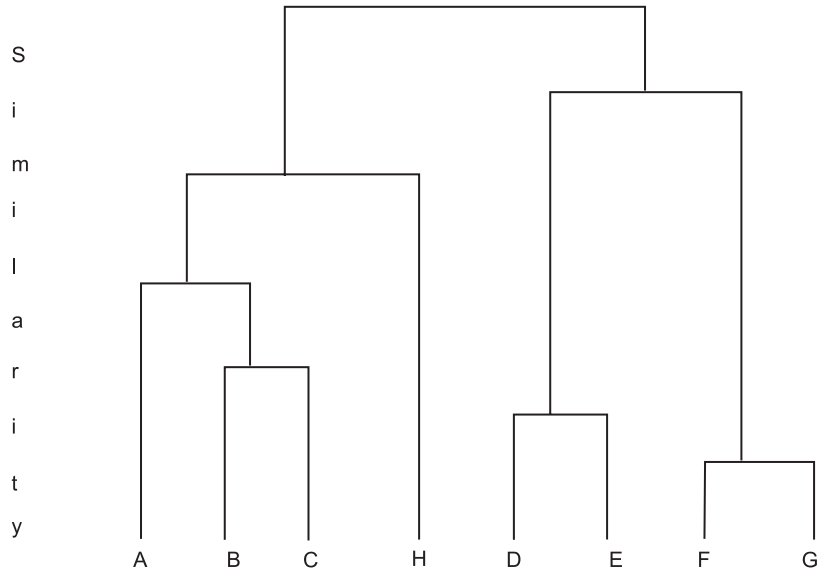


Figure 9.10 Dendrogram for a polythetic clustering of the eight 2-dimensional points

At the next level, we partition the cluster {A, B, C, H} into {A, B, C} and {H} and at the subsequent level, the cluster {A, B, C} is split into two clusters: {A}, and {B, C}; now we have 5 clusters given by {A}, {B, C}, {H}, {D, E}, and {F, G}. Similarly,

a dendrogram depicts partitions having 6, 7 and 8 clusters of the data at successive levels. Observe that at the final level, each cluster has only one point; such clusters are singleton clusters.

Here, to obtain the optimal 2-partition of a cluster of size n, $2^n - 1$ 2-partitions are to be considered and the best of them is chosen; so, $O(2^n)$ effort is required to generate all possible 2-partitions and select the most appropriate partition.

It is possible to use one feature at a time to partition (monothetic clustering) the given data set. In such a case, a feature direction is considered and the data is partitioned into two clusters based on the gap in the projected values along the feature direction. That is, the data set is split into two parts at a point that corresponds to the mean value of the maximum gap found among the values of the data's feature. Each of these clusters is further partitioned sequentially using the remaining features.

EXAMPLE 7

Monothetic clustering can be illustrated using Figure 9.11.

Here, we have 8 two-dimensional patterns; x_1 and x_2 are the two features used to describe these patterns. Taking feature x_1, the data is split into two clusters based on the maximum inter-pattern gap which occurred between two successive patterns in the x_1 direction. If we consider the x_1 values of the patterns in increasing order, they are

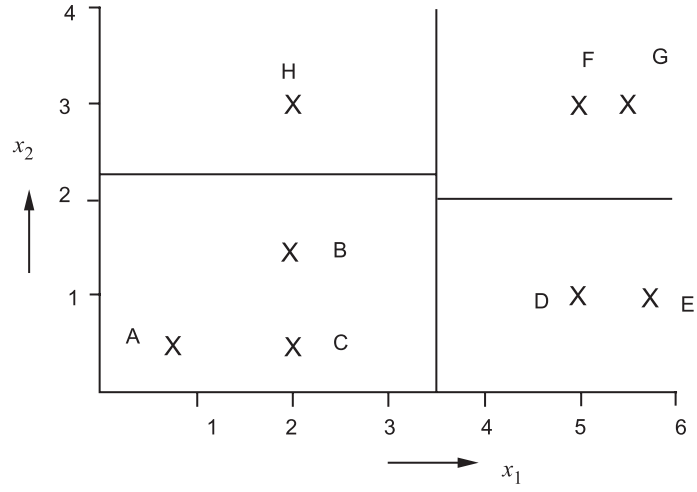


Figure 9.11 Monothetic divisive clustering

A: 0.5, B: 2, H: 2, C: 2, D: 5, F: 5, G: 5.5, and E: 5.75

So, the maximum inter-pattern gap ($5 - 2 = 3$ units) occurs between C and D. We

select the mid-point between 2 and 5 which is 3.5 and use it to split the data into two clusters. They are

C_1 = {A, B, C, H} and C_2 = {D, E, F, G}

Each of these clusters is split further into two clusters based on the value of x_2; the split again depends on the maximum inter-pattern gap between two successive patterns in the x_2 direction. Ordering the patterns in C_1 based on their x_2 values, we get

A: 0.5, C: 0.5, B: 1.5, and H: 3.0

Here, the maximum gap of 1.5 units $(3 - 1.5 = 1.5$ units) occurs between B and H. So, by splitting C_1 at the mid-point, 2.25, of the feature dimension x_2, we get two clusters

C_{11} = {H}, and C_{12} = {A, B, C}

Similarly, by splitting C_2 using the value of 2 for x_2, we get

C_{21} = {F, G}, and C_{22} = {D, E}

A major problem with this approach is that in the worst case, the initial data set is split into 2^d clusters by considering all the d features. This many clusters may not be typically acceptable, so an additional merging phase is required. In such a phase, a pair of clusters is selected based on some notion of nearness between the two clusters and they are merged into a single cluster. This process is repeated till the clusters are reduced to the required number. There could be different ways of characterising the nearness between a pair of clusters—distance between the centroids of the two clusters is the popularly used characteristic.

Sorting the n elements in the data and finding the maximum inter-pattern gap is of $O(n \log n)$ time complexity for each feature direction and for the d features under consideration, this effort is $O(dn \log n)$. So, this scheme can be infeasible for large values of n and d.

9.2.2 Agglomerative Clustering

Typically, an agglomerative clustering algorithm goes through the following steps:

STEP 1: Compute the similarity/dissimilarity matrix between all pairs of patterns. Initialise each cluster with a distinct pattern.

STEP 2: Find the closest pair of clusters and merge them. Update the proximity matrix to reflect the merge.

STEP 3: If all the patterns are in one cluster, stop. Else, goto Step 2.

Note that Step 1 in the above algorithm requires $O(n^2)$ time to compute pair-wise similarities and $O(n^2)$ space to store the values, when there are n patterns in the given collection (refer to Exercise 12). There are several possible variations to store the similarity matrix. One approach is to place the matrix either in the main memory or on a disc. Another possibility is to store only the patterns and compute the similarity values based on need. The former requires more space while the latter requires more time.

There are several ways of implementing the second step. In the single-link algorithm, the distance between two clusters C_1 and C_2 is the $minimum$ of the distances $d(X, Y)$, where $X \in C_1$ and $Y \in C_2$. In the complete-link algorithm, the distance between two clusters C_1 and C_2 is the $maximum$ of the distances $d(X, Y)$,where $X \in C_1$ and $Y \in C_2$.

EXAMPLE 8

The single-link algorithm can be explained with the help of the data shown in Figure 9.9. The dendrogram corresponding to the single-link algorithm is shown in Figure 9.12.

Note that there are 8 clusters to start with, where each cluster has one element. The distance matrix using city-block distance or Manhattan distance is given in Table 9.9.

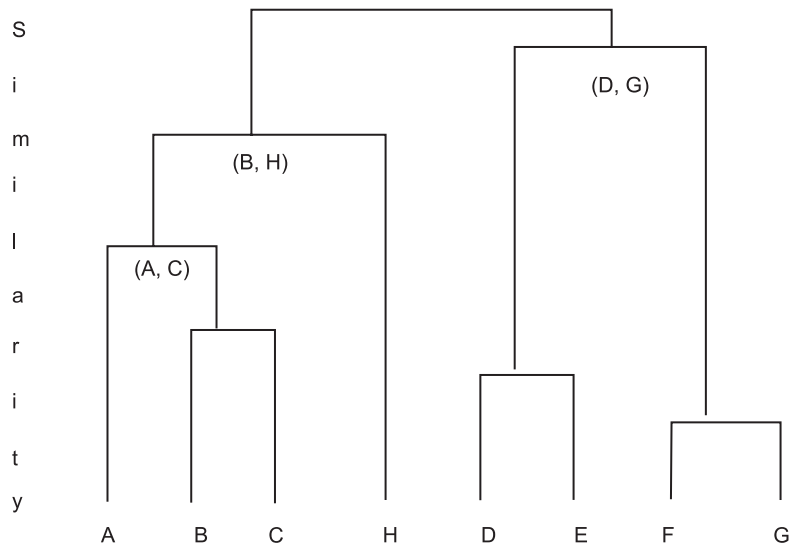


Figure 9.12 Dendrogram for the single-link algorithm

Table 9.9 The ijth entry in the matrix is the city-block distance between X_i and X_j. It is defined as $d(X_i, X_j) = \sum_{i=1}^{l} \mid X_{il} - X_{jl} \mid$, where l is the number of dimensions, here $l = 2$.

	A	B	C	D	E	F	G	H
A	0	2.5	1.5	5	5.75	7	7.5	4
B	2.5	0	1.0	3.5	4.25	4.5	5	1.5
C	1.5	1	0	3.5	4.25	5.5	6	2.5
D	5	3.5	3.5	0	0.75	2	2.5	5
E	5.75	4,25	4.25	0.75	0	2.75	2.25	5.75
F	7	4.5	5.5	2	2.75	0	0.5	3
G	7.5	5	6	2.5	2.5	0.5	0	3.5
H	4	1.5	2.5	2.5	5.75	3	3.5	0

Since the clusters {F} and {G} are the closest to each other with a distance of 0.5 units, they are merged to realise a partition of 7 clusters. The updated matrix after merging {F} and {G} into one cluster, {F, G} is given in Table 9.10

Clusters {D} and {E} are the next closest with a distance of 0.75 units and they are merged next. In a similar fashion, clusters {B} and {C} are next merged. Then the cluster comprising B and C is merged with the cluster {A}. This cluster is then merged with the cluster {H}. Next the cluster consisting of {D, E} and the cluster consisting of {F, G} are merged. At this stage, there are two clusters. The process can be stopped whenever we have the number of clusters we want. The dendrogram in Figure 9.12 shows the merging of clusters at various levels. Here, clusters {A} and {B, C} are merged based on the nearness between A and C. Similarly, {A, B, C} and {H} are merged based on the distance between B and H.

Table 9.10 The updated similarity matrix after one merger

	A	B	C	D	E	F,G	H
A	0	2.5	1.5	5	5.75	7	4
B	2.5	0	1.0	3.5	4.25	4.5	1.5
C	1.5	1	0	3.5	4.25	5.5	2.5
D	5	3.5	3.5	0	0.75	2	5
E	5.75	4,25	4.25	0.75	0	2.25	5.75
F, G	7	4.5	5.5	2	2.75	0	3
H	4	1.5	2.5	2.5	5.75	3	0

Applying the complete-link algorithm on Figure 9.9, the dendrogram generated by the complete-link algorithm is shown in Figure 9.13.

Here, clusters {A} and {B,C} are merged based on the nearness between A and B. Similarly, {A, B, C} and {H} are merged based on the distance between A and H.

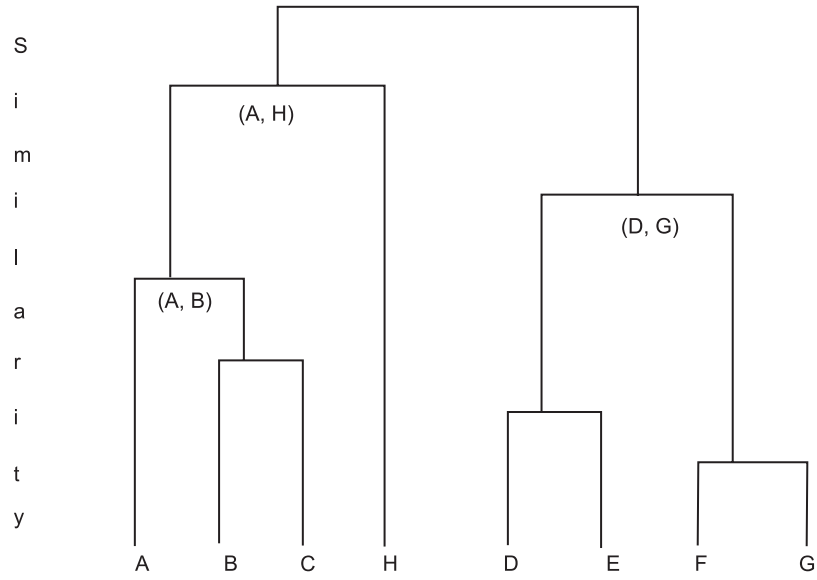


Figure 9.13 Dendrogram for the complete-link algorithm

It is easy to observe that the complete-link algorithm generates compact clusters as it relates every pattern in a cluster with every other pattern in the cluster. On the other hand, the single-link algorithm (SLA) characterises the presence of a pattern in a cluster, based on its nearest neighbour in the cluster. The positive aspect is that the SLA is highly versatile and can generate clusters of different shapes. For example, the SLA can generate the 2-partition of the data set shown in Figure 9.14. Note that there are two-concentric clusters in the data.

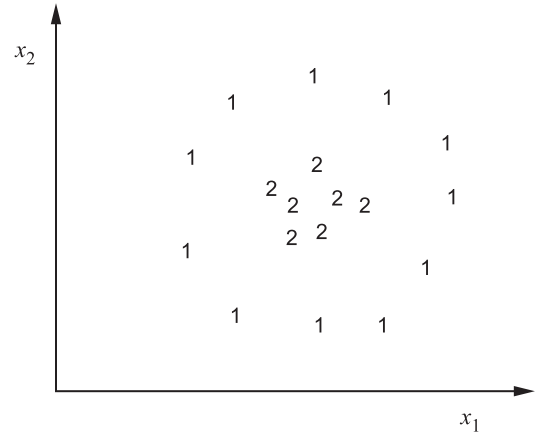


Figure 9.14 Concentric clusters

Hierarchical clustering algorithms are computationally expensive. The agglomerative algorithms require computation and storage of a similarity or dissimilarity matrix of values that has $O(n^2)$ time and space requirement. They can be used in applications where hundreds of patterns were to be clustered. However, when the data sets are larger in size, these algorithms are not feasible because of the non-linear time and space demands. It may not be easy to visualise a dendrogram corresponding to 1000 patterns. Similarly, divisive algorithms require exponential time to analyse the number of patterns or the number of features. So, they too do not scale up well in the context of large-scale problems involving millions of patterns.

9.3 Partitional Clustering

Partitional clustering algorithms generate a hard or soft partition of the data. The most popular of this category of algorithms is the *k-means* algorithm.

9.3.1 k-Means Algorithm

A simple description of the k-means algorithm is given below.

STEP 1: Select k out of the given n patterns as the initial cluster centres. Assign each of the remaining $n-k$ patterns to one of the k clusters; a pattern is assigned to its closest centre/cluster.

STEP 2: Compute the cluster centres based on the current assignment of patterns.

STEP 3: Assign each of the n patterns to its closest centre/cluster.

STEP 4: If there is no change in the assignment of patterns to clusters during two successive iterations, then stop; else, goto Step 2.

There are a variety of schemes for selecting the initial cluster centres; these include selecting the first k of the given n patterns, selecting k random patterns out of the given n patterns, and viewing the initial cluster seed selection as an optimisation problem, and using a robust tool to search for the globally optimal solution. Selecting the initial cluster centres is a very important issue.

EXAMPLE 9

k-means algorithm may be illustrated with the help of the two-dimensional data set of 7 points shown in Figure 9.15 and Figure 9.16.

The same collection of points is used in both the figures. In each case, they are clustered to generate a 3-partition $(k = 3)$. The patterns are located at A = (1, 1), B = (1, 2), C = (2, 2), D = (6, 2), E = (7, 2), F = (6, 6), G = (7, 6) If A, D, and F are selected as the initial centres as shown in Figure 9.15, Cluster 1 has $(1, 1)$

as its cluster centre, Cluster 2 has $(6, 2)$ as its cluster centre and Cluster 3 has $(6, 6)$ as its cluster centre. The patterns B, C, E and G are assigned to the cluster which is closest to them. B and C are assigned to Cluster 1; E is assigned to Cluster 2; and G is assigned to Cluster 3. The new cluster centre of Cluster 1 will be the mean of the patterns in Cluster 1(mean of A, B and C) which will be $(1.33, 1.66)$. The cluster centre of Cluster 2 will be $(6.5, 4)$ and the cluster centre of Cluster 3 will be $(6.5, 6)$. The patterns are again assigned to the closest cluster depending on the distance from the cluster centres. Now, A, B and C are assigned to Cluster 1, D and E are assigned to Cluster 2 and F and G are assigned to Cluster 3. Since there is no change in the clusters formed, this is the final set of clusters.

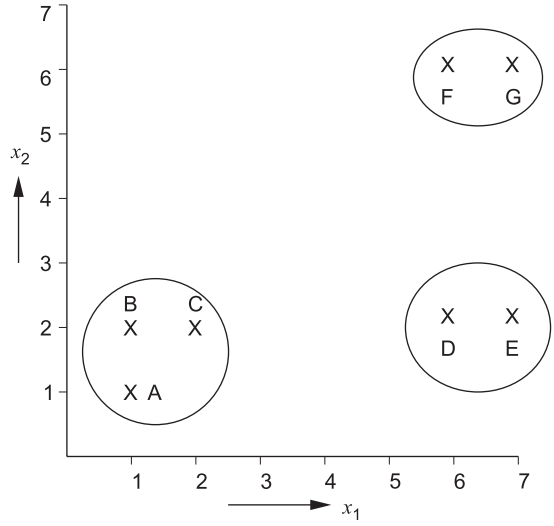


Figure 9.15 Optimal partition when A, D and F are the initial means

This gives a visually appealing partition of three clusters—the clusters {A, B, C}, {D, E}, {F, G} are generated. On the other hand, by starting with A, B, and C as the initial centres, we end up with the partition shown in Figure 9.16 which seems to have smaller variances in two clusters and a large variance in the third. Note that the variance in each cluster is acceptable in the partition shown in Figure 9.15.

An important property of the k-means algorithm is that it implicitly minimises the sum of squared deviations of patterns in a cluster from the centre. More formally, if C_i is the ith cluster and μ_i is its centre, then the criterion function minimised by the algorithm is

$$\sum_{i=1}^{k} \sum_{x \in C_i} (x - \mu_i)^t (x - \mu_i)$$

In clustering literature, it is called the *sum-of-squared-error criterion*, *within-group-error sum-of-squares*, or simply *squared-error criterion*. Note that the value of the function is 2.44 for the partition in Figure 9.15, whereas its value is 17.5 for the partition in Figure 9.16. This shows that the k-means algorithm does not guarantee the globally optimal partition.

A frequently used heuristic is to select the initial k centres so that they are as far away from each other as possible; this scheme is observed to work well in practice. For example, the selection made in Figure 9.15 is one such. Note that the selection of A, E, and G as initial centres results in the same partition as the one shown in Figure 9.15.

The time complexity of the algorithm is $O(nkdl)$, where l is the number of iterations and d is the dimensionality. The space requirement is $O(kd)$. These features make the algorithm very attractive. It is one of the most frequently used algorithms in a variety of applications. Some of these applications involve large volumes of data, for example, satellite image data. It is best to use the k-means algorithm when the clusters are hyper-spherical. It does not generate the intended partition if the partition has non-spherical clusters, for example, the concentric clusters shown in Figure 9.14.. Even when the clusters are spherical, it may not find the globally optimal partitions as depicted in Figure 9.16. Several schemes have been used to find the globally optimal partition corresponding to the minimum value of the squared-error criterion. We discuss them in the next section.

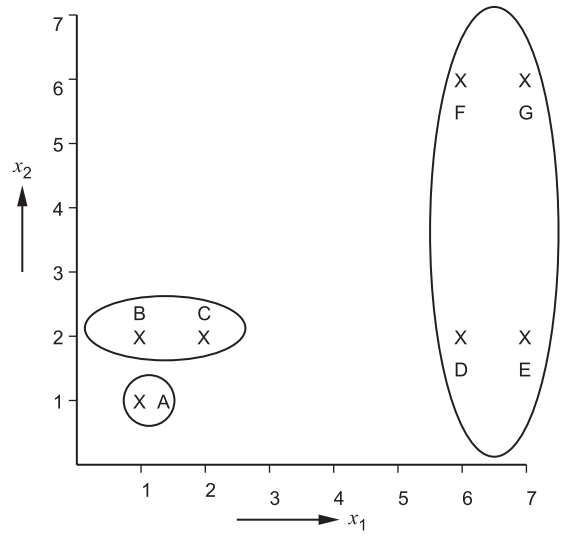


Figure 9.16 A non-optimal partition when A, B and C are the initial means

9.3.2 Soft Partitioning

The conventional k-means algorithm employs the *winner-take-all* strategy (that is, assign a pattern to the winning cluster only) and generates a hard partition. A major reason for it to generate a locally optimal solution is poor initial seed/cluster centre selection. Some of the solutions that were provided in literature for this problem are:

1. *Use a split and merge strategy* Here, clusters generated by the k-means algorithm are considered for possible splitting and merging. If the sample variance of a cluster is more than a user-specified threshold value, T_v, then the cluster is split into two clusters by choosing the most dissimilar pair of patterns in the cluster as the initial seeds. Similarly, if the distance between the centres of two clusters is less than a distance threshold, T_d, then they are merged into one cluster. This *split and merge* strategy is used by the iterative self-organising data analysis technique (ISODATA). Note that using appropriate values for T_v and T_d, it is possible to generate the optimal partition shown in Figure 9.15 from the partition shown in Figure 9.16.

   This is achieved by selecting the cluster {D, E, F, G} for splitting based on the variance going beyond the user-specified threshold T_v. By selecting as initial centroids either D or E and either F or G, we get the two clusters {D, E} and {F, G}. Also, by selecting the clusters {A} and {B, C} for merging as the distance between A and the centroid of the cluster {B, C} is less than a user-defined threshold T_d and merging them results in the partition shown in Figure 9.15.

   However, the difficulty in implementing this strategy is in estimating the appropriate values for the threshold parameters T_v and T_d (refer to Exercise16).

2. *Use a soft competitive learning strategy* For example, use the *winner-take-most* approach instead of the *winner-take-all* method. This means that along with the centre that is closest to the given pattern, some of the other neighbouring centres also get affected by the pattern. The pattern has varying degrees of effect on these centroids. The nearest centre is affected more than the others. This method has been shown to decrease the dependency on the initialisation. It is implemented by the following soft clustering approaches.

   (a) *Fuzzy clustering*: Here, each pattern is assigned to different clusters (typically, more than one cluster) based on a membership value. Its value in a cluster is computed using the pattern and the corresponding cluster centroid.

   (b) *Rough clustering*: Here, each cluster is assumed to have a non-overlapping part and an overlapping part. Patterns in the non-overlapping portion of a cluster belong to only that cluster. Patterns in the overlapping part may belong to two or more clusters.

   (c) *Neural networks based clustering*: Here, soft competitive learning is used to obtain a soft partition.

3. *Use a stochastic search technique* They ensure that the clusters converge to a globally optimal partition in a probabilistic manner. The most popular techniques that are used in clustering are:

   (a) *Simulated annealing* Here, the current solution is updated randomly and the resulting solution is accepted with a probability value. If the resulting solution is better than the current solution, it is accepted; otherwise, it is accepted with a probability in the range 0 to 1.

   (b) *Tabu search* Here, unlike the simulated annealing scheme, more than one solution is stored and the current solution is perturbed in more than one way to find the next configuration.

   (c) *Evolutionary algorithms* Here, a population of solutions is maintained. In addition to the fitness values of the individuals, a random search based on interaction among solutions with mutation is used to generate the next population. Genetic algorithms, evolutionary search and evolutionary programming are different schemes coming under this approach. All these approaches are used in solving the optimal clustering problem.

9.4 Clustering Large Data Sets

An objective way of characterising largeness of a data set is by specifying bounds on the number of patterns and features present. For example, a data set having more than a billion patterns and/or more than a million features is *large*. However, such a characterisation is not universally acceptable; further, this characterisation may change with developments in technology. So, it may be good to look for a pragmatic characterisation. A data set is large if it is not possible to fit the data in the main memory of the machine on which it is processed. The data then resides in a secondary storage device and has to be transferred to the main memory based on need. Accessing the secondary storage space is several orders slower than accessing the main memory. This assumption is behind the design of various data mining tasks where large data sets are routinely processed. As a result, two algorithms may have the same time complexity, say $O(n)$, where n is the number of patterns to be clustered, but it is possible that one of them is feasible and the other is not. For example, an incremental algorithm that scans the data set once to generate the partition requires $O(n)$ time and the k-means algorithm (KMA) may scan the data set several times, say l times, to reach the final partition. The time complexity of KMA is $O(n)$ for fixed values of k and l. So, both the algorithms have the same time complexity; however, if the value of l is 100, then KMA has to scan the data set 100 times, whereas the incremental algorithm scans the data set only once. So, it is possible that on large data sets, KMA may not generate the partition in an acceptable time whereas the incremental algorithm can do so.

The above discussion motivates us to use the *number of data set scans* as a parameter to assess the performance of clustering algorithms on large data sets. In this chapter, we use the number of data set scans to compare clustering algorithms regarding their pragmatic behaviour. Also, we use the O notation to rule out the feasibility of some of the algorithms as follows:

1. Algorithms having non-linear time and space complexities are not feasible.
2. Even linear time and space algorithms may not be feasible; they tend to be unfeasible as the number of data set scans increases.

9.4.1 Possible Solutions

The following solutions have been offered in literature:

1. *Use an incremental clustering algorithm* It clusters a new pattern based on the existing cluster representatives and some threshold parameters. It does not reconsider patterns seen earlier. Therefore, it generates cluster representatives with the help of a single data set scan. Algorithms like *leader*, described in the next section, employ incremental clustering strategies. It is possible to use the incremental clustering algorithm in both *off-line*(where the entire data set is available to start with) and *on-line* (where the data is made available incrementally) applications.
2. *Generate an intermediate abstraction and use it* Here, the data is transferred from the disc and an abstraction of the data is generated. This abstraction is used for clustering. The abstraction may be generated by examining the data incrementally or non-incrementally.
   (a) *Generate the intermediate representation incrementally* Scan the data set once or twice and construct a representation that occupies much lesser space than the original data—in this way, the representation generated may fit in the main memory. Use this representation, and not the original data, for clustering.
   (b) *Generate the intermediate abstraction non-incrementally and use the it for clustering* The most popular scheme under this category is the *divide-and-conquer* approach. Transfer a part of the data at a time into the main memory and cluster the data in the main memory. Store the cluster representatives . Repeat this process as many times as is needed based on the size of the entire data residing on the disc and the size of the data transferred into the main memory each time. This is a two-level algorithm. It is possible to have a multi-level divide-and-conquer approach based on the sizes of the data and the main memory.

We will discuss these approaches in the next two sub-sections.

9.4.2 Incremental Clustering

Incremental clustering is based on the assumption that it is possible to consider patterns one at a time and assign them to existing clusters. A new data item is assigned to a cluster without affecting the existing clusters significantly. A detailed description of a typical incremental clustering algorithm is given below:

An Incremental Clustering Algorithm (Leader Algorithm)

STEP 1: Assign the first data item, P_1, to cluster C_1; set $i = 1$ and $j = 1$ and let the leader L_i be P_j.

STEP 2: Set $j = j + 1$; consider clusters C_1 to C_i in the increasing order of the index and assign P_j to cluster C_m $(1 \leq m \leq i)$ if the distance between L_m and P_j is less than a user-specified threshold T; if no cluster satisfies this property, then set $i = i + 1$ and assign P_j to the new cluster C_i. Set the leader L_i to be P_j.

STEP 3: Repeat Step 2 till all the data items are assigned to clusters.

An important feature of the above incremental algorithm, called *leader algorithm*, is that it only requires a *single database scan*. Further, it needs to store only the cluster representatives in memory. So, it has the best time and space complexities.

Order-independence is an important property of clustering algorithms. An algorithm is *order-independent algorithm* if it generates the same partition for any order in which the data is presented. Otherwise it is *order-dependent*. Unfortunately, the incremental algorithm mentioned above is order-dependent.

EXAMPLE 10

Consider the data given in Figure 9.15. If the data is processed in the order A, B, C, D, E, F and G, and if the threshold T is 3, A is assigned to C_1 and A is the leader of C_1. Then B and C are assigned to C_1 since the distance from B and C to A is below the threshold 3. D is assigned to a new cluster C_2 and D is the leader of this cluster. E is assigned to cluster 2. The distance of F from A (the leader of C_1) is 7.07 and from D (the leader of C_2) is 4. Both these distances are above the threshold and therefore F is put into the new cluster C_3 and is made the leader of this cluster. G is assigned to C_3.

It can be seen that if the data had been processed in a different order, the cluster leaders would have been different—even the clusters can vary. If C occurred before A and B, then C would have been the leader of C_1. If D occurred before and the distance between C and D was less than the threshold, it would have been in C_1. This may not occur if A is the leader. The leader algorithm is therefore order dependent.

9.4.3 Divide-and-Conquer Approach

A major difficulty in clustering large data is that it is not possible to store the entire data set under consideration in the main memory. A natural way of solving this problem is to consider a part of the data set at a time and store the corresponding cluster representatives in the memory. Specifically, consider that we have stored data of size $n \times d$ in a secondary storage space. We divide this data into p blocks, where an optimal value of p can be chosen based on the algorithm used. We transfer each of these blocks to the main memory and cluster it into k clusters using a standard clustering algorithm. If we represent each of these clusters by a representative, then we will have pk representatives corresponding to k clusters by considering all the p blocks. These pk representatives are clustered into k clusters and the cluster labels of these representative patterns are used to re-label the original pattern matrix. We depict this two-level algorithm in Figure 9.17.

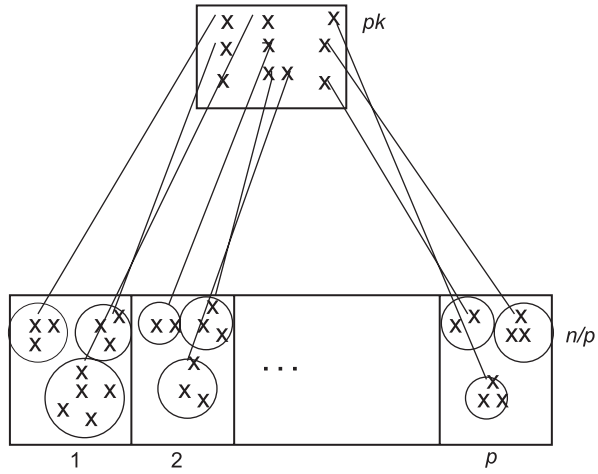


Figure 9.17 Divide-and-conquer approach to clustering

EXAMPLE 11

This example is done for illustrative purposes. Generally, the divide-and-conquer-method is carried out only for large data sets. Consider the set of patterns

$$
\begin{array}{llll}
\mathrm{X}_1 = (1,1); & \mathrm{X}_2 = (2,1); & \mathrm{X}_3 = (1,1.5); & \mathrm{X}_4 = (1,2); \\
\mathrm{X}_5 = (1,3); & \mathrm{X}_6 = (2,3); & \mathrm{X}_7 = (2,2); & \mathrm{X}_8 = (3,2); \\
\mathrm{X}_9 = (4,1); & \mathrm{X}_{10} = (5,1); & \mathrm{X}_{11} = (5,2); & \mathrm{X}_{12} = (6,1); \\
\mathrm{X}_{13} = (6,2); & \mathrm{X}_{14} = (6,3); & \mathrm{X}_{15} = (5,3); & \mathrm{X}_{16} = (5,3.5); \\
\mathrm{X}_{17} = (4,4); & \mathrm{X}_{18} = (5,4); & \mathrm{X}_{19} = (4,5); & \mathrm{X}_{20} = (6,4); \\
\mathrm{X}_{21} = (5,5); & \mathrm{X}_{22} = (6,5); & \mathrm{X}_{23} = (5,6); & \mathrm{X}_{24} = (6,6).
\end{array}
$$

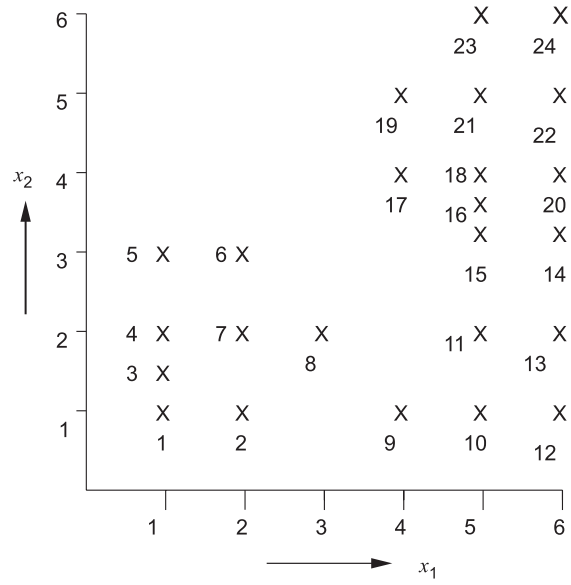


Figure 9.18 Example data set for divide-and-conquer strategy

This is shown in Figure 9.18. If the sub-set of patterns 1 to 8 is considered as one block, sub-set of patterns 9 to 16 as the second block and the remaining set of patterns 17 to 24 as the third block, then each block has to be clustered separately. In the first block, three clusters are formed namely, one consisting of patterns 1, 2 and 3 with the centroid at $C_1 = (1.33, 1.17)$; the second consisting of patterns 4, 5 and 6 with the centroid at $C_2 = (1.33, 2.67)$; and the third consisting of patterns 7 and 8 with centroid at $C_3 = (2.5, 2)$. In the second block, three clusters are formed namely, one consisting of patterns 9,10 and 11 with the centroid at $C_4 = (4.67, 1.33)$; the second consisting of patterns 12, 13 and 14 with the centroid at $C_5 = (6, 2)$; and the third consisting of patterns 15 and 16 with the centroid at $C_6 = (5, 3.25)$. In the third block, three clusters are formed namely, one consisting of patterns 17, 18 and 19 with the centroid at $C_7 = (4.33, 4.33)$; the second consisting of patterns 20, 21 and 22 with the centroid at $C_8 = (5.66, 4.66)$; and the third consisting of patterns 23 and 24 with the centroid at $C_9 = (5.5, 6)$.

These 9 centroids which are calculated, form patterns in the next level which have to be clustered. These patterns used for clustering at the second level are shown in Figure 9.19. Depending on the clustering done in the second level, the patterns in the first level are also labelled as belonging to a cluster to which the centroid of its cluster belongs. If C_1, C_2 and C_3 are clustered together to form Cluster 1, C_4 and C_5 are clustered together to form Cluster 2 and C_6, C_7, C_8 and C_9 are clustered together to form Cluster 3, then patterns 1, 2, 3, 4, 5, 6, 7 and 8 will be in Cluster 1, patterns

9, 10, 11, 12, 13 and 14 will be in Cluster 2 and patterns 15, 16, 17, 18, 19, 20, 21, 22, 23 and 24 will be in Cluster 3.

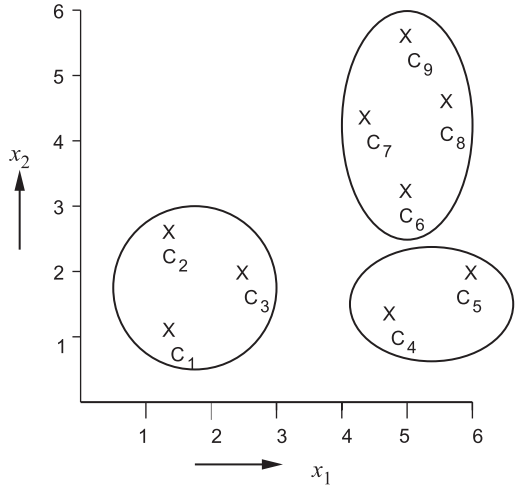


Figure 9.19 Clustering carried out at level 2 using divide-and-conquer strategy

It is possible to extend this approach in different ways. Some of them are listed below:

1. A multi-level divide-and-conquer approach can be used where the number of levels is more than 2. More levels are required if the data set size is very large and the main memory size is small.
2. Centroids can be used as representatives at any level.
3. It is possible to use different clustering algorithms at different levels. This view leads to the hybrid clustering approaches. It is possible to use the k-prototypes algorithm at the first level and obtain the representatives; and then use a hierarchical algorithm on these representatives. This is the hybrid approach, where a computationally efficient partitioning algorithm is used at the first level to achieve data compression and the resulting set of centroids is clustered using a versatile algorithm like the single-link algorithm (SLA). Even though SLA is expensive, in this case, it has to cluster only the first level representatives and not the entire data set. This is depicted in Figure 9.20.

An abstraction where a cluster of points is represented by their centroid helps in data compression as discussed earlier and the representatives can be used for further clustering or as prototypes for classification.

Use of the centroid to represent a cluster is the most popular scheme. It works well when the clusters are compact or isotropic. However, when the clusters are elongated or non-isotropic, this scheme fails to represent them properly. This is shown in Figure 9.21. Figure 9.21(a) shows two clusters which have an identical centroid. Figure 9.21(b) shows clusters which are chain-like and where the centroid may not be a good representative. Centroid-based algorithms are not robust as shown earlier.

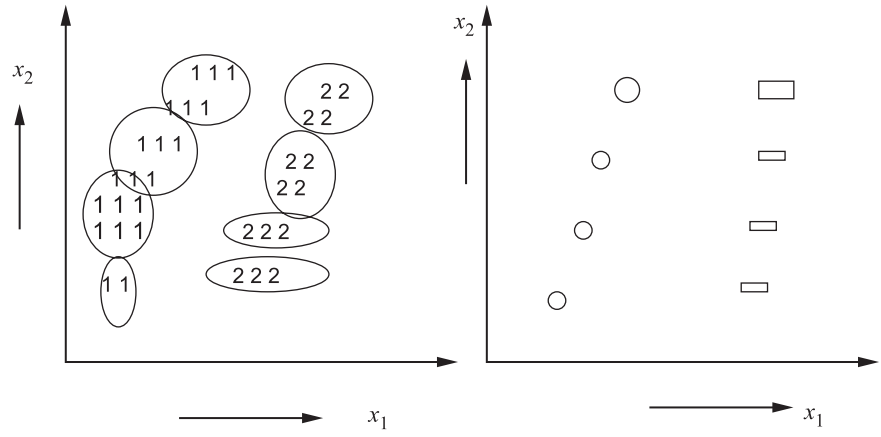


Figure 9.20 Hybrid clustering

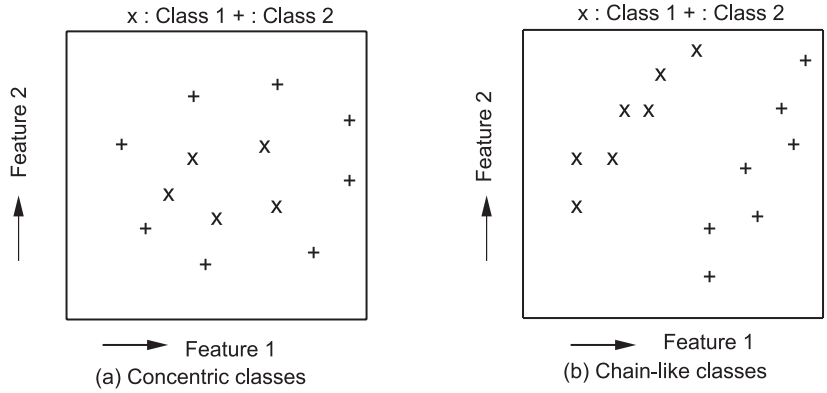


Figure 9.21 Example of data sets having concentric and chain-like classes

Discussion

Clustering is an important tool in pattern recognition. It is not an end product in itself but is useful in several decision-making situations including classification. Clustering may be seen as a compression tool; it generates an abstraction of the data and often

it is easier to deal with such an abstraction rather than the original data itself. In this chapter, we have described several important steps in the process of clustering and explained some popularly used clustering algorithms.

Further Reading

Duda et al. (2000) have discussed clustering techniques. A review of different clustering methods has been done by Jain et al. (1999). Filippone et al. (2008) have compiled a survey of some clustering methods. The k-means algorithm for clustering is discussed by Bagirov (2008), Chang et al. (2009) and Likas et al. (2003).

Clustering using soft computing techniques is explained by Klein and Dubes (1989), Al Sultan (1995), Gancarski et al. (2008), Chang et al. (2009), Liu et al. (2009), Lughofer (2008) and Spath (1980).

Asharaf and Murty (2003), Bicego and Figueiredo (2009) and Wu and Yang (2002) discuss clustering using fuzzy and rough set techniques. Xiong and Yeung (2004) discuss time series data. Some other papers on clustering have been written by Chan et al. (2004), Dougherty and Brun (2004), Xiang et al. (2008) and Yousri et al. (2009).

Exercises

1. Consider the following 10 patterns.
   $X_1 = (1, 1)$ $X_2 = (6, 1)$ $X_3 = (2, 1)$ $X_4 = (6, 7)$
   $X_5 = (1, 2)$ $X_6 = (7, 1)$ $X_7 = (7, 7)$ $X_8 = (2, 2)$
   $X_9 = (6, 2)$ $X_{10} = (7, 6)$
   Obtain the distance matrix using the squared Euclidean distance as the distance between two points.
2. If there is a set of n patterns and it is required to cluster these patterns to form two clusters, how many such partitions will there be?
3. Show that using all the four object types in Table 9.3, there are at most 16 predicates. List them.
4. Show that for the cluster of 5 points shown in Example 2, the medoid m is (1.6,1.4).
5. Show that the medoid m^2 specified in Example 2 is (2, 2).
6. Consider Example 7 where 8 patterns are depicted in Figure 9.11. Obtain an optimal 3-partition (partition with 3 clusters) of the set {A, B, C, H}.

7. Consider the hierarchical clustering of 8 patterns shown in Figure 9.11 using the monothetic clustering algorithm. Obtain the corresponding dendrogram.

8. Show that in the worst case, a data set of n d-dimensional patterns is split into 2^d clusters ($n > 2^d$) using a monothetic hierarchical clustering algorithm.

9. Show that for clustering n d-dimensional patterns using the monothetic clustering algorithm, the effort required to obtain the feature direction with the maximum inter-pattern gap is $O(dn \log n)$. Here, effort is required to sort the patterns in each of the d dimensions. Find the maximum inter-pattern gap in each of these dimensions and select the maximum of the d gaps found.

10. Consider the set of two-dimensional patterns :

    (1, 1), (1, 2), (2, 1), (2, 1.5), (3, 2), (4, 1.5), (4, 2), (5, 1.5),
    (4.5, 2), (4, 4),(4.5, 4),(4.5, 5), (4, 5), (5, 5)

    Use divisive clustering (both monothetic and polythetic) and agglomerative (single-link and complete-link) clustering to obtain

    (a) 3 clusters
    (b) 4 clusters

11. Consider the two-dimensional data set given below:

    (3, 3), (3, 4), (4, 3), (4, 4), (1, 2), (1, 3), (1, 4), (1, 5), (2, 6), (3, 7), (4, 7),
    (5, 7), (6, 6), (7, 5), (7, 4), (7, 3), (7, 2), (6, 1), (5, 1), (4, 1), (3, 1), (2, 1).

    Use the single-link algorithm and the complete-link algorithm to cluster these points into two clusters. Do you get the same 2-partition of the data set using both these algorithms? What can you infer?

12. Show that the computation of the proximity (similarity/distance) matrix for n patterns requires $\frac{n(n-1)}{2}$ distance computations ($O(n^2)$ time) and storing of as many values. You may assume that the distance function is symmetric, which means $d(i, j) = d(j, i)$.

13. For the patterns in Exercise 10, use k-means algorithm where

    (a) $k = 3$
    (b) $k = 4$

14. Consider the collection of three-dimensional patterns: (1, 1, 1), (1, 2, 1), (1, 1, 2), (6, 6, 1), (6, 7, 1), (7, 6, 1). Find the 2-partition obtained by the k-means algorithm with $k = 2$. What is the value of the error-sum criterion?

15. Consider the collection of two-dimensional patterns :

    (1, 1, 1), (1, 2, 1), (2, 1, 1), (2, 1.5, 1), (3, 2, 1), (4, 1.5, 2), (4, 2, 2),
    (5, 1.5, 2), (4.5, 2, 2), (4, 4, 3), (4.5, 4, 3), (4.5, 5, 3), (4, 5, 3), (5, 5, 3)

    where each pattern is represented by feature 1, feature 2 and the class. Find the centroid and medoid of each class.

16. Consider the data given in Example 9 and the discussion on splitting and merging k-means clusters based on T_v and T_d in Section 9.3.2. Work out the appropriate values for T_v and T_d for the data in Example 9 so that it is possible to get clusters shown in Figure 9.15 from the partition shown in Figure 9.16.

17. Use the incremental clustering algorithm for the patterns in Exercise 2. Explore the use of different thresholds.

18. Give an example to illustrate the order-dependence of the leader algorithm.

19. Give an example to illustrate that the centroid of a cluster may not be its best representative.

Computer Exercises

1. Consider the data provided in Example 1 for clustering based on a threshold of 5 units on the squared Euclidean distance. Design an algorithm and implement it to generate the 3-partition shown in the example.

2. Consider running the NN on cluster representatives rather than the original data points. Using the 16 labelled patterns in Example 3, and clustering them using your code in the previous exercise to have 4 cluster representatives, compare the time taken to classify a test pattern using all the 16 patterns as the training data versus using only the 4 cluster centroids as the training data. Use 80 test patterns, 40 from the range-box (1, 1) to (2, 6) (so the test pattern will have its x_1 value between 1 and 2 and its x_2 value between 1 and 6) and the remaining 40 may be chosen in the range-box (6, 1) to (7, 7).

3. Implement the single-link algorithm. Record the time taken to compute the proximiy matrix, and generate clustering. Use the data set shown in Example 11.

4. Implement the k-means algorithm and study the impact of initial seed selection on the resulting k-partition. You may consider the data set shown in Example 9.

5. Compare the versatility of the single-link algorithm with that of the k-means algorithm by running the code developed in the previous two exercises on the data set shown in Exercise 11.

Bibliography

1. Al-Sultan, K. S. A Tabu search approach to the clustering problem. *Pattern Recognition* 28: 1443–1451. 1995.
2. Asharaf, S., M. Narasimha Murty. An adaptive rough fuzzy single pass algorithm for clustering large data sets. *Pattern Recognition* 36(12): 3015–3018.
3. Bagirov, Adil M. Modified global k-means algorithm for minimum sum-of-squares clustering problems. *Pattern Recognition* 41(10): 3192–3199. 2008.
4. Bicego, Manuele, Mario A.T. Figueiredo. Soft clustering using weighted one-class support vector machines. *Pattern Recognition* 42(1): 27–32. 2009.
5. Chan, Elaine Y., Wai Ki Ching, Michael K. Ng, Joshua Z. Huang. An optimization algorithm for clustering using weighted dissimilarity measures. *Pattern Recognition* 37(5): 943–952. 2004.
6. Chang, Dong-Xia, Xian-Da Zhang and Chang-Wen Zheng. A genetic algorithm with gene rearrangement for k-means clustering. *Pattern Recognition* 42(7): 1210–1222. 2009.
7. Dougherty, Edward R., Marcel Brun. A probabilistic theory of clustering. *Pattern Recognition* 37(5): 917–925.
8. Duda, R.O., P.E. Hart, D.G. Stork. *Pattern Classification*. John Wiley and Sons. 2000.
9. Filippone, Maurizio, Francesco Camastra, Francesco Masulli, Stefano Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recognition* 41(1): 176–190. 2008.
10. Gançarski, P., A. Blansché, A. Wania. Comparison between two co-evolutionary feature weighting algorithms in clustering. *Pattern Recognition* 41(3): 983–994. 2008.
11. Jain, A.K. M. Narasimha Murty, P.J. Flynn. cata Clustering: A Review. *ACM Computing Surveys* 31(3): 264–329. 1999.
12. Klein, R.W. and R.C. Dubes. Experiments in projection and clustering by simulated annealing. *Pattern Recognition* 22: 213–220. 1989.
13. Likas, Aristidis, Nikos Vlassis, Jakob J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition* 36(2): 451–461. 2003.

14. Liu, Manhua, Xudong Jiang, Alex C. Kot. A multi-prototype clustering algorithm. *Pattern Recognition* 42(5): 689–698. 2009.

15. Lughofer, Edwin. Extensions of vector quantization for incremental clustering. *Pattern Recognition* 41(3): 995–1011. 2008.

16. Spath, H. *Cluster Analysis: Algorithms for Data Reduction and Classification of Objects*. West Sussex, U. K.: Ellis Horwood. 1980.

17. Wu, Kuo-Lung, Miin-Shen Yang. Alternative c-means clustering algorithms. *Pattern Recognition* 35(10): 2267–2278. 2002.

18. Xiang, Shiming, Feiping Nie, Changshui Zhang. Learning a Mahalanobis distance metric for data clustering and classification. *Pattern Recognition* 41(12): 3600–3612. 2008.

19. Xiong, Yimin, Dit-Yan Yeung. Time series clustering with ARMA mixtures. *Pattern Recognition* 37(8): 1675–1689. 2004.

20. Yousri, Noha A., Mohamed S. Kamel, Mohamed A. Ismai. A distance-relatedness dynamic model for clustering high dimensional data of arbitrary shapes and densities. *Pattern Recognition* 42(7): 1193–1209. 2009.

10 Summary

Pattern recognition (PR) is an important and mature area of scientific activity. The two important activities under pattern recognition are pattern classification and clustering. We have discussed in detail several classification and clustering algorithms in the previous chapters. Our main emphasis was on elaborating the concept using an algorithmic approach. Such a treatment is useful while applying the tools in real life. Some theoretical details have been sacrificed to make this possible but several simple examples have been provided to help the reader understand the concepts.

The most important step in pattern recognition is the representation of both patterns and classes. An exhaustive and easy to understand coverage has been provided here. After which we considered specific recognition algorithms. It is important to understand that there is no general theory for representing patterns in any domain. However, a good representation scheme helps in achieving better classifiers. Unfortunately, in literature, the importance given to this step is insignificant. We have tried to collect several aspects and structure them to encourage the reader to appreciate the difficulties involved.

Nearest neighbour classifiers are the most popular in literature as they are very simple for both the human and the machine (no learning) and at the same time are robust. Due to this reason, we have covered them in detail. A major difficulty associated with them is the classification time. There is no design time; however, classification (testing) time is linear in the number of training patterns. In order to improve the situation, several efficient pre-processing schemes have been proposed in literature. They employ either a reduced data set or a data structure and an associated efficient algorithm to find the nearest neighbours. We have discussed several efficient algorithms.

Bayes classifier is an answer to those patterns which require an optimal classifier. It has a strong theoretical flavour of probability and statistics. However, estimation of the associated probability distributions could be an arduous task in practice. The naive Bayes classifier is an efficient implementation of the overall approach. We have discussed both the Bayes classifier and naive Bayes classifier with appropriate examples to illustrate the concept.

Hidden Markov models are popular in speech and speaker recognition tasks. They

can be used to characterise classes of patterns where each pattern is viewed as a sequence of sub-patterns or as a sequence of states. We have tried to explain HMMs with simple examples to illustrate the concepts. We have also discussed the use of hidden Markov models for pattern classification.

Decision trees are friendly data structures to both the programmer and the manager (decision maker). They are frequently used in both pattern classification and data mining. The decision tree can deal with both numerical and categorical features. We have covered the axis parallel decision tree based classifiers in detail with appropriate examples. There are other categories of decision trees including oblique decision trees. They are computationally expensive but we have covered them briefly and the interested reader may look up the references for more details.

Linear discriminant function based classifiers have played an important role in the recent development of pattern recognition. The perceptron is one of the earliest classifiers which has been studied both from analytical and practical angles. It led to the development of artificial neural networks, specifically, multilayer perceptrons. A more recent exploit, in this area, is the support vector machine (SVM). SVM can be easily identified as the most successful and popular classifier in the past decade. Statistical pattern recognition gained a significant prominent place because of SVMs. These algorithms have been considered along with suitable examples.

Another prominent direction in pattern recognition is the use of more than one classifier to decide the class label of a test pattern. There are several possible schemes for combining classifiers. An important contribution here is the ADABOOST algorithm. We have covered various schemes for implementing combinational classifiers.

Clustering is an important tool in recognition. Even though it is studied as an end product in itself, it is not always so in practice. Typically, the results of clustering are useful for further decision making. In such a context, clustering may be viewed as an abstraction generation tool—for example, pre-processing of the training data to reduce its size. Hierarchical and partitioning schemes for clustering form the dichotomy of the different approaches and we have discussed both these categories in detail.

Most of the important classification and clustering algorithms have been discussed in the text. However, the reader may benefit by referring to the appropriate publications which are cited in a structured manner in the references section at the end of each chapter.

11 An Application: Handwritten Digit Recognition

In this chapter, we present an example of how pattern classification can be carried out in a digit recognition problem. There are ten classes corresponding to the handwritten digits ''0'' to ''9''. The data set consists of 6670 training patterns and 3333 test patterns. The nearest neighbour algorithm (NN), the kNN and the mkNN algorithms have been used on this data set. Classification is carried out for the test patterns and the classification accuracy reported.

To overcome the problem of using the nearest neighbour algorithm on such a large data set, it is first condensed. The k-means algorithm (KMA) and the fuzzy c-means algorithm (FCMA) are used on the data and the centroids of the clusters formed are used as prototypes representing all the patterns in the cluster. The reduced set so formed is used to obtain the classification accuracy of the test set. Condensation of the training patterns has been carried out using the MCNN algorithm and the CNN algorithm. This reduced set has been used on the test data to obtain classification accuracy. Even though the current application deals with a digit recognition problem, it is possible to extend the scheme to other pattern recognition applications dealing with large data sets.

11.1 Description of the Digit Data

Each original digit pattern is a binary image of size 32×24 pixels. So, the dimensionality of the original pattern is 768 (32×24). Keeping the computational resources we have in mind, the dimensionality of the data set is reduced as follows. Non-overlapping windows of size 2×2 are formed over the entire image and each window is replaced by one feature whose value corresponds to the number of one bits in that window. This results in 192 features, where the value of each feature varies from 0 to 4. There are 6670 (667×10) training patterns and 3333 test patterns. Figure 11.1 shows some of the patterns in the training data set. It can be seen that the patterns in the data vary in terms of orientation of the digit, width and height.

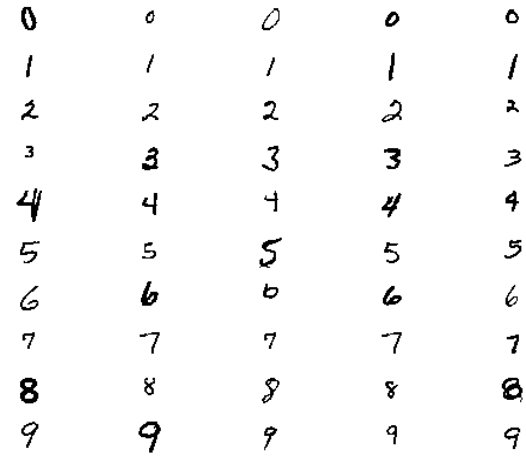


Figure 11.1 A sample collection of training patterns

To understand the data better, central tendency and dispersion of the patterns is considered. To do this, the non-zero features of the binary pattern is taken. Table 11.1 provides the results.

It can be seen that in the case of Class 0, the patterns consist of features ranging from 39 to 121. A high standard deviation reflects the large variation in the number of non-zero features of the patterns of the class.

Table 11.1 Statistics on non-zero features of the data set

Class Id.	Arithmetic mean of 1s per pattern	Standard deviation	Actual min. and max. of 1s across all patterns
0	66.8	12.2	(39, 121)
1	29.0	5.9	(16, 54)
2	63.8	11.5	(34, 105)
3	59.0	11.7	(27, 109)
4	52.2	9.4	(26, 96)
5	60.7	11.3	(31, 110)
6	57.7	9.4	(35, 94)
7	46.5	8.7	(26, 113)
8	66.7	12.9	(36, 117)
9	54.8	9.1	(31, 88)

The dimensionality of the data set is further reduced. Non-overlapping windows of size 2×2 is formed over the 16×12 digit data and each window is replaced by one value corresponding to the sum of the four values falling in the window. This reduces each digit to a 8×6 matrix of values in the range 0 to 16. These values are

linearly mapped to the range 0 to 4 and each pattern gets a total of 48 features. This reduction does not affect the shape of the character significantly and was necessitated to overcome the curse of the dimensionality problem and also to improve the speed of the nearest neighbour type of algorithms.

11.2 Pre-processing of Data

Pre-processing of data is to be carried out to bring about some uniformity in the data and make it amenable to classification.

The three most important operations to pre-process a character image are: (1) scaling, (2) translation, and (3) rotation. The first two operations are reasonably easy to perform and so they are performed frequently to normalise the image size. Pre-processing is done based on translation and scaling as described below.

1. *Translation* The maximum size of a character is 192 (16×12) pixels where (1,1) corresponds to the first row and first column of the character. Similarly, (16,12) corresponds to the last row (16th row) and last column (12th column). So, the column index varies between 1 and 12 and the row index varies from 1 to 16. The character is shifted so that its centre in the horizontal direction falls on the 6th column which lies half way between columns 1 and 12 and its centre in the vertical direction falls on the eighth row which is half way between rows 1 and 16.
2. *Scaling* The pattern is stretched in the vertical direction. It is cut horizontally into two halves. The top half is shifted to start on the first row and the bottom half ends on the 16th row. Intermediate rows, if empty, are filled based on their nearest non-empty rows.

11.3 Classification Algorithms

A variety of neighbourhood classifiers have been used on the digit recognition data. The class label of a test pattern is decided based on the label of one or more neighbours here. The specific classifiers we have used are the nearest neighbour classifier (NN), the k-nearest neighbour classifier (kNN), and modified k-nearest classifier (MkNN). These algorithms have been explained in detail in Chapter 3.

11.4 Selection of Representative Patterns

Neighbourhood classifiers work well. For example, it was observed, based on a large collection of experiments, that the kNN is the most robust classifier among the

statistical pattern classifiers. However, it tends to use the entire training data set to obtain the neighbours of a test pattern. So, it has a time complexity of $O(\text{NMK})$, where N is the number of training patterns, M is the number of test patterns, and K is the number of neighbours obtained. There are several classifiers used earlier which use a representative set or a sub-set of the training data set. These include the minimal distance classifier (MDC), and the condensed nearest neighbour classifier (CNN). These methods have been explained in detail in Chapter 3.

Obtaining the condensed data set is a time-consuming process but once it is available, classification can be much faster as compared to using the entire training data set.

One method is to cluster the data and use the cluster centres as the prototypes in the condensed set. The popular algorithm for clustering is the k-means algorithm (KMA). Making use of the concept of fuzzy membership, we use the fuzzy c-means algorithm (FCMA).

CNN is typically good at reducing the size of the training data set. But this is achieved at the cost of some reduction in the classification accuracy. Also, CNN is *order-dependent*. The obtained *condensed* set varies in size and contents based on the order in which the training patterns are presented to the above algorithm.

To develop an order-independent algorithm which ideally gives the optimal set of patterns in the condensed set, an algorithm was developed which gives the same set of patterns even if the order of the data set is changed. In other words, it is an order-independent algorithm. This is the MCNN algorithm described in Chapter 3.

11.5 Results

Several experiments were conducted on the digit recognition data using the nearest neighbour based classifiers. Two training sets were with: (1) all the 6670 training patterns, and (2) a condensed set of 1715 training patterns obtained by using the CNNC on all the training patterns. Further, in each of these cases, we worked with two values for the dimensionality of the data: (1) using all the 192 features corresponding to the data set described earlier, and (2) using only 48 features obtained by further feature reduction. These results are presented in Table 11.2 and Table 11.3. For each case, the classification accuracy is calculated. This is a measure of how many patterns are classified correctly using the particular classification algorithm. If $N1$ patterns out of N patterns are classified correctly, then the classification accuracy, CA is

$$\text{CA} = \frac{N1}{N} \times 100$$

It is possible to observe from these tables that the mkNN is performing better than the other two classifiers and the best classification accuracy of 92.57% is obtained

by using mkNN on the entire training data set with 192 features as reported in Table 11.2.

Both the raw data and the pre-processed data have been used for the neighbourhood classifiers. The pre-processed data gives better results.

When centroids of classes are used as training patterns—a requirement in MDC—it is necessary to pre-process the data as described earlier. This data has been considered using all the 192 features and also using only 48 features. The results obtained using these data sets are reported in Table 11.4 and Table 11.5. It can be observed from these tables that MDC performs much better on the pre-processed data. This is expected because the centroid will represent processed data patterns better than the raw patterns. Due to this, only pre-processed data has been used to obtain representatives (centroids) using clustering algorithms KMA and FCMA.

Table 11.2 NN results using 192 features

Number of training patterns	NN(%) accuracy	kNN(%) accuracy	mkNNC(%) accuracy
6670 (all)	91.50	91.60	**92.57**
6670 (pre-proc)	92.17	92.50	**93.10**
1715 (condensed)	86.20	87.19	90.60

Table 11.3 NNC results using 48 features

Number of training patterns	NN(%) accuracy	kNN(%) accuracy	mkNN(%) accuracy
6670 (all)	85.00	86.20	88.18
6670 (pre-proc)	92.71	92.62	**93.49**
1715 (condensed)	76.33	81.55	82.84

Table 11.4 MDC results using 192 features

Number of training patterns	Raw data accuracy(%)	Pre-processed data accuracy(%)
6670 (all)	68.5	79.48
1715 (condensed)	68.08	80.89

Table 11.5 MDC results using 48 features

Number of training patterns	Raw data accuracy (%)	Pre-processed data accuracy (%)
6670 (all)	66.20	80.26
1715 (condensed)	65.92	81.55

KMA and FCMA were used to find out the centroids that represent various classes. Two sets of experiments were conducted corresponding to different number of centroids. In the first case, each class of 667 training patterns is grouped into 50 clusters and each of the resulting clusters is represented by the centroid of the data in the cluster. As a result, the entire collection of 6670 training patterns is represented by 500 centroids, 50 centroids from each class. In the second case, each class of training patterns is represented by 150 centroids after grouping the patterns into 150 clusters. This approach results in a total of 1500 centroids from the ten classes corresponding to the digits ''0'' to ''9''. In each case, clustering is done using KMA and FCMA. The classification accuracy is computed using NN, kNN, and mkNN on all the test patterns with the set of centroids forming the training data. We present the results obtained in Table 11.6 and Table 11.7.

It may be observed from both Table 11.6 and Table 11.7 that using KMA to obtain the centroids and having a small number of centroids gives results almost as good as using the entire training data. The FCMA takes a lot of time as compared to the KMA.

Besides, NNC, kNN and mkNN, there is the fuzzy kNN algorithm (FKNN). This has been used for classifying the test data using the centroids obtained by KMA and FCMA. These results are given in Table 11.8.

Table 11.6 Results using 500 centroids as training data

Clustering algorithm used	NN accuracy (%)	kNN ($k = 5$) accuracy (%)	mkNN ($k = 5$) accuracy (%)
KMA	92.41	90.28	**93.10**
FCMA	85.39	85.15	85.72

Table 11.7 Results using 1500 centroids as training data

Clustering algorithm used	NN accuracy (%)	kNN ($k = 5$) accuracy (%)	mkNN ($k = 5$) accuracy (%)
KMA	93.00	92.83	**93.64**
FCMA	88.69	88.12	88.87

Table 11.8 Results using fuzzy kNN classifier

Clustering algorithm used	No. of centroids	Accuracy (%)
None	6670	93.13
KMA	500	92.71
KMA	1500	93.94
FCMA	500	85.06
FCMA	1500	88.75

The results show that the accuracy obtained is of the same magnitude as that obtained using NN, kNN and mkNN.

The MCNN is an order-independent algorithm which finds a condensed set of prototypes to represent the training set. A set of patterns were found using MCNN and these patterns were used to carry out classification of the test data using NN algorithm. The results are presented in Table 11.9 and are compared to the CNN algorithm.

Table 11.9 Classification accuracy using CNN and MCNN

Algorithm used	No. of prototypes	NN algorithm accuracy (%)
CNN	1580	87.04 %
MCNN	1527	88 %
All patterns	6670	92 %

Another method of clustering is the leader algorithm explained in Chapter 8. This is an incremental algorithm which goes through the training data set only once and is therefore suitable for large data sets which are stored in secondary storage devices. In this method, the threshold is fixed and the resulting number of clusters depend on this threshold. Table 11.10 gives the number of prototypes and the classification accuracy obtained by using different threshold values. As the threshold is increased, the number of clusters reduces. This results in fewer prototypes. The classification accuracy reduces as the threshold is increased (i.e., when the number of prototypes decrease).

Table 11.10 Results using leader algorithm for clustering

Distance threshold	No.of prototypes	C.A. %
5	6149	91.24
10	5581	91.24
15	4399	90.40
18	3564	90.20
20	3057	88.03
22	2542	87.04
25	1892	84.88
27	1526	81.70

Discussion

In this chapter, we discussed the application of pattern classification algorithms to a digit-recognition problem. The classification algorithms used were the nearest neighbour and related algorithms. Results were observed for data obtained by reducing the number of features to 192 and 48. Besides, results obtained using raw data and pre-processed data were also noted.

The training data were also reduced by some methods and the results observed. The first of these methods was MDC which uses the centroid of each class as a representative pattern of the class. This does not give a good classification accuracy. The centroids obtained by the KMA and FCMA have also been used to classify the test patterns. The CNN and the MCNN algorithms have been used to obtain a condensed set which has been used for classification. MCNN is better as it is an order-independent algorithm. The leader algorithm is another attractive algorithm for clustering data. The clusters obtained depend on the threshold used. Various threshold values have been tried out. Each cluster is represented by the cluster centre or leader which is used as a prototype in the NN algorithm.

Further Reading

Devi and Murty (2000) have used NN, kNN, mkNN, CNN, FCNN, KMA and MDC on digit recognition data and observed the results. The results of using MCNN and CNN on the same digit recognition data are also given by Devi and Murty (2002).

Bibliography

1. Devi, V. Susheela and M. Narasimha Murty. An incremental prototype set building technique. *Pattern Recognition* 35: 505–513. 2002.
2. Devi, V. Susheela and M. Narasimha Murty. Handwritten digit recognition using soft computing tools. In *Soft Computing for Image Processing*. Edited by S. K. Pal, A. Ghosh and M. K. Kundu. Berlin: Springer. 2000.

Index

χ^2 distribution, 139
χ^2 pruning, 139
χ^2 table, 139

abstraction, 2, 24
activation function, 174
ADABOOST algorithm, 192, 246
adenine, 9
agglomerative algorithms, 221, 222
agriculture, 3
alpha-numerical characters, 1
alphabet, 13
apriori, 86
artificial
    intelligence, 2
    neural networks, 41, 169, 221, 246
    neurons, 169
association rule mining, 55
attributes, 8
audio data, 1
automatic
    auditing of records, 220
    medical diagnosis, 1
    recognition, 26
average probability of error, 88
axis-parallel test, 125
axon, 169

back-propagation, 172
    learning algorithm, 41
background knowledge, 87
bagging, 190
Bayes
    classifier, 4, 86
    theorem, 86
Bayesian
    network, 97
    scheme, 90
belief network, 97
benchmark, 4
best individual features, 36
between class scatter matrix, 27
binary
    chromosomes, 79
    classifier, 147
    pattern matrix, 219
    valued vector, 41
binomial distribution, 91
bioinformatics, 3, 104, 220
biomedical data analysis, 220
biometrics, 3, 220
bit string, 41
Boolean OR, 165
booster, 192
boosting, 192
bootstrap
    aggregation, 190
    procedure, 43
    replicate, 190
bottom-up approach, 36
bottom-up strategy, 222
bounding boxes, 11
branch and bound
    algorithm, 56
    search, 33
    technique, 31

categorical features, 4, 127, 246
central tendency, 248
centrally located pattern, 24
centroid, 16
chain-like classes, 69, 239
character recognition, 3
chromosomal DNA, 9
chromosome, 41
city-block distance, 226
class
    conditional feature distributions, 96
    conditional probabilities, 88
    label, 2
    membership, 53
    priors, 95
classification, 1
    error, 32
    problem, 8
    rule, 2
    time, 42
    tools, 65
classifiers
    combination of, 188
    complexity, 31
    ensemble of, 188
    imperfect, 188
cluster
    identity, 2
    representatives, 24, 208, 234
    seed selection, 229
    tendency, 220
clustering, 2, 207
clusters, 207
cohesive groups, 207
combinational classifiers, 246
combinations of classifiers, 5
common prefix, 13
complete-link algorithm, 226
compressed tree, 13
compressed tree structure, 13
computational
    burden, 2, 8, 66
    complexity, 220
    efficiency, 31
    requirements, 62, 219
concentric
    classes, 239
    clusters, 231
conceptual
    cohesiveness, 22
    distance, 22
condensation algorithms, 68
condensed nearest neighbour (CNN)
    algorithms, 69
    classifier, 250
conditional
    dependencies, 97
    independence, 94
    probability table, 98
confusion, 75
conjunction of logical disjunctions, 9
constrained optimisation problem, 179
context, 20
continuous random variables, 91
correlated variables, 29
covariance matrix, 17
crisp set, 10
criterion function, 27, 32
cross-validation, 43, 138
cube algorithm, 58
curse of dimensionality, 31
cytosine, 9

data
    re-organisation, 219
    reduction, 65
    set scans, number of, 234
    space, 23
    structure, 4
data mining, 220
    tools, 2
database, 12

decision
    making, 2, 221
    rule, 48
    stump, 192
    tree, 4, 123
degree of membership, 53
Delauney tree (DT), 10
deletion, 11
dendrogram, 221
descendants, 11
design time, 42, 65, 129
detection of intrusions, 220
dichotomise, 129
difference vector, 18
diffusion, 75
digit recognition, 113
dimensionality reduction, 31
directed acyclic graphs, 97
discrete-valued pattern, 25
discriminating
    attributes, 8
    axis, 64
discriminative attribute, 131
dispersion of the patterns, 248
dissimilar predicates, 22
dissimilarity measures, 16
distance
    matrix, 209
    measures, 4, 16
    weighted k-nearest neighbour algorithm, 51
divide-and-conquer
    approach, 234, 236
    strategy, 220
divisive algorithms, 221, 222
DNA sequences, 9, 104
DNA sub-sequences, 104
document
    analysis, 220
    recognition, 3, 220
    retrieval, 23
domain knowledge, 2, 22
dynamic Bayesian networks, 97

edges in the tree, 11
edit distance, 19
edited prototype set, 75
education, 3
efficient algorithms, 55
electrochemical signals, 169
entertainment, 3
entropy impurity, 132
error correcting output coding, 196
estimation of probabilities, 90
Euclidean distance, 16
evolutionary
    algorithms, 221, 233
    programming, 233
    search, 233
exclusive OR, 173, 175
exhaustive search, 31–33
expert systems, 3
explanation ability, 42
extra-logical evidence, 214
extra-logical information, 213

face, 2
fault diagnostics, 3
feature, 3
    extraction, 26
    selection, 31, 66, 79
feed-forward network, 171
finance, 3
fingerprints, 1, 2
first-order Markov assumption, 106
first-order Markovian property, 108
Fisher's linear discriminant, 26
fitness function, 79
formal language theory, 4
Fourier transform, 1
FP growth algorithm, 13
frequent item sets, 13, 25
frequent pattern (FP) tree, 11

fuzziness, 10
fuzzy
  kNN algorithm, 53
  c-means algorithm (FCMA), 247
  clustering, 232
  label, 10
  numbers, 10
  pattern, 10
  sets, 4, 10, 221

gating
  function, 201
  network, 201
Gaussian distribution, 73, 91
generalisation, 22
generalised eigenvectors, 27
genetic algorithms (GA), 41, 221, 233
Gini impurity, 133
globally optimal solution, 34
grades of membership, 10
gradient descent approach, 174
graph neighbours, 79
guanine, 9

Hamming distance, 114
hand-held devices, 42
hard class labels, 10
hard clustering algorithms, 211, 221
Hausdorff distance, 18
header node, 15
hidden
  layer, 174
  Markov models (HMMs), 4, 103, 111
hierarchical algorithms, 221
hierarchical schemes for clustering, 246
high density data regions, 77
high probabilities of error, 31
holdout method, 43
homogeneous form, 149, 152
human brain, 169
hybrid
  approach, 238
  clustering, 239
hyper-sphere, 64
hyper-spherical, 231
hypercube, 58

image data, 26
imprecise data, 10
incremental algorithm, 233
incremental nearest neighbour search, 64
indexing scheme, 220
inducer, 192
influence diagrams, 98
information
  impurity, 132
  retrieval, 220
injecting randomness, 197
insertion, 11
inter-cluster distance, 209
inter-pattern gap, 225
intermediate
  abstraction, 234
  representation, 234
internal node, 123
intervals, 10
intra-cluster distance, 209
irrelevant features, 127
iterative condensation algorithms (ICA), 69
iterative self-organising data analysis technique (ISODATA), 232

joint distribution, 98
joint probability model, 94

k-d tree, 10, 64
k-fold cross-validation, 43
k-means algorithm (KMA), 229, 247
k-median distance, 18
k-nearest neighbour (kNN)
  algorithm, 50
  classifier, 24, 249
kernel functions, 22

keywords, 1, 23
Kullback–Leibler distance (KL distance), 19

L_1 distance, 16
L_2 distance, 16
labelled
  clustering, 218
  patterns, 167
Lagrange variables, 179
large
  data sets, 4
  databases, 13
  samples, 48
  training data set, 5
leader, 234
  algorithm, 235
leaf node, 123
learning
  algorithms, 188
  from examples, 2
  from observations, 2
least salient nodes, 41
least significant feature, 37
leave-one-out approach, 43
Levenshtein distance, 19
likelihood values, 86, 91
linear
  combination of features, 125
  decision boundaries, 5, 149
  discriminant functions, 147, 246
linearly separable, 148
linguistic
  tools, 4
  values, 10
links, 11
local distribution, 98
logical
  connectives, 213
  description, 9
lower
  approximation, 10
  bound, 56
  dimensional space, 29

machine learning, 2
Mahalanobis distance, 17
majority class, 54
Manhattan distance, 16, 226
manipulating
  the input features, 196
  the output targets, 196
MAP decision rule, 96
market-basket data, 54
Markov chain, 199
  Monte Carlo (MCMC) method, 198
Markov process, 199
maximum a posterior decision rule, 96
maximum likelihood
  estimation, 90
  scheme, 90
maximum variance, 29
max–min approach, 39
mean, 24
mean-square error, 30
measures of impurity, 132
medical diagnostics, 42
medicine, 3
medoid, 24
membership function, 53
metric, 16
minimal distance classifier (MDC), 66, 250
minimum
  bounding rectangles, 11
  distance classifier(MDC), 24
  spanning tree (MST), 10, 11
  support, 55
  support value, 55
minimum error rate
  classification, 4
  classifier, 88
Minkowski metric, 16
misclassification impurity, 133
missing values, 10

modified k-nearest neighbour (MkNN)
algorithm, 51
classifier, 24, 249
modified branch and bound algorithm (BAB$^+$), 34
modified condensed nearest neighbour (MCNN) algorithms, 69
monothetic clustering, 222
monotonicity, 33
most significant feature, 37
multi-dimensional space, 4, 147, 149
multi-level divide-and-conquer approach, 238
MULTIEDIT, 75
multilayer
feed-forward network, 41
perceptrons, 173, 246
multimedia
data, 1
data analytics, 3
document recognition (MDR), 1
multiple representatives, 24
multivariate split, 138
music, 1
mutual neighbourhood
distance (MND), 20
value, 79

naive Bayes classifier, 93
nearest neighbour (NN)
algorithm, 48
classifier, 24, 249
rule, 4, 48
nearest neighbour based classifiers, 48
negative half
portion, 151
space, 149
nested sequence of partitions, 221
nesting effect, 36
neural networks, 4
based clustering, 232
neurons, 169
nitrogenous bases, 9
noise tolerance, 42
noisy
data, 4
patterns, 112
non-homogeneous form, 149
non-leaf node, 11, 123
non-linear
combination of features, 125
decision boundaries, 173
non-linearly separable data, 167
non-metric, 16
non-rectangular regions, 129
non-spherical clusters, 231
normalisation, 153
of data, 23
normalised probabilities, 115
numeric data, 2
numerical features, 4, 127, 246

object recognition, 3
oblique decision trees, 246
oblique split, 137
observation symbol, 114
off-line applications, 234
on-line applications, 234
optimal classifier, 86
optimality, 4
optimum error, 48
order-dependent, 71
algorithm, 235
order-independent, 74, 235
ordered
database, 14
orthogonal basis, 29
partitions, 62
orthogonal basis, 29
outcome, 123
outlier, 54
overfitting, 111, 139

parity function, 131

partition of the data, 2
partitional algorithms, 221
partitional clustering algorithms, 229
partitioning schemes for clustering, 246
pattern recognition, 1
patterns
  as strings, 9
  as trees and graphs, 10
  as vectors, 8
  labelled, 208
  unlabelled, 208
peaking phenomena, 31
perceptron, 5, 246
  learning algorithm, 153, 154
physical object, 7
physical systems, 54
pixel, 13
"plus l take away r" selection, 37
polynomial kernel, 22
polythetic clustering, 222
positive
  half space, 149, 152
  reflexivity, 16
posterior probabilities, 86
pre-defined concepts, 22
pre-processing
  of data, 249
  techniques, 2, 62
precise statements, 10
prediction, 216
  rule, 192
principal component, 29
  analysis (PCA), 29
prior probabilities, 86
probabilistic
  classifier, 93
  dependencies, 97
  graphical model, 97
probability
  density, 4
  density function, 91
  distributions, 4, 19
  model, 93
  of error, 48
  of misclassification, 31
  of the hypothesis, 86
probability and statistics, 245
projection, 4
  algorithms, 60
  space, 27
protein sequence, 9
prototype selection, 66, 218
prototypical pattern, 5
proximity, 2
  matrix, 225
  measures, 16
pruning, 41, 139

quick sort algorithm, 220

r near neighbours, 54
R-tree, 10
radial basis function kernel, 23
random sub-sampling, 43
random variable, 29
raw data, 1
RBF kernel, 23
real numbers, 10
recurrence relation, 20
relaxed branch and bound (RBAB), 34
remote-sensed data analysis, 220
representation, 2
  of clusters, 15
  of patterns, 8
representative, 15
  groups, 56
  patterns, 68, 249
root node, 14
rotation, 249
rough
  clustering, 232
  sets, 4, 10, 221
  values, 10

rule-based systems, 2

saliency of a node, 41
sample sizes, 31
sample variances, 222
scaling, 249
scientific experiments, 54
search procedures, 31
searching, 11
security, 3
semantic, 2
sentence in a language, 9
sequence of
    images, 1
    states, 246
    sub-patterns, 246
sequential backward selection (SBS), 36, 37
sequential floating
    backward search (SBFS), 39
    forward search (SFFS), 38
    search, 37
sequential forward selection (SFS), 36
sequential selection, 36
set of
    classes, 8
    concepts, 22
    objects, 8
sigmoid function, 174
signal processing, 1
signature, 1
significant feature, 37
similar predicates, 22
similarity, 2
    function, 2
    measures, 16
simulated annealing, 79, 233
simultaneous editing, 79
single database scan, 235
single-link algorithm, 226
soft clustering algorithms, 221
soft competitive learning, 232
soft computing tools, 4
soft partition, 211
space requirements, 2, 66
spanning tree, 11
speaker
    recognition, 3
    recognition tasks, 245
speech, 1, 2
    recognition, 3, 4
    tasks, 245
split and merge strategy, 232
squared distance, 62
squared Euclidean distance, 209
squared-error criterion, 231
stacking, 201
state
    sequences, 111
    transition diagrams, 106
    transition matrix, 108
stationarity assumption, 107
statistical pattern recognition, 4, 246
statistics and probability, 4
stochastic
    search technique, 41
    techniques, 79
sub-classes, 5
sub-optimal procedures, 31
sub-sampling training examples, 190
sub-spaces, 4
sum-of-squared-error criterion, 231
supervised learning, 2, 24
support, 14
    lines, 176
    planes, 176
    vector machines (SVMs), 5, 147, 246
    vectors, 25, 147, 176
symmetric matrix, 29
symmetry, 16
synapses, 169
syntactic, 2
syntactic pattern recognition, 4

Tabu search, 79, 233
target recognition, 220
telecommunications networks, 54
theorem of the ugly duckling, 213
threshold, 14
thymine, 9
time requirements, 2, 66
top-down
    approach, 37
    strategy, 222
training
    phase, 4
    set, 16
transaction databases, 12
transactions at a supermarket, 54
transformation matrix, 27
transition probability matrix, 107
translation, 249
transportation, 3
triangular inequality, 16
truth table, 173
typical pattern, 78

uncertain values, 10
uncertainty, 4
uncorrelated variables, 29
unlabelled patterns, 2
unsupervised
    classification, 214
    learning, 16
updation, 11
upper approximation, 10

validation set, 79
variance impurity, 132
vector
    representation, 8
    spaces, 4
vectors of numbers, 2
video data, 1
Voronoi regions, 72

weak hypothesis, 192
weight vector, 154
weighted
    distance measure, 17
    Euclidean distance, 17
    majority rule, 51
    vote, 199
    voting methods, 200
winner-take-all strategy, 232
winner-take-most, 232
within class scatter matrix, 27
within-group-error sum-of-squares, 231

XOR, 175

zero vector, 150