ADVANCES IN LARGE MARGIN CLASSIFIERS

EDITED BY

ALEXANDER J. SMOLA

PETER L. BARTLETT

BERNHARD SCHÖLKOPF

DALE SCHUURMANS

Advances in Large Margin Classifiers

Advances in Neural Information Processing Systems

Published by Morgan-Kaufmann

NIPS-1
Advances in Neural Information Processing Systems 1: Proceedings of the 1988 Conference,
David S. Touretzky, ed., 1989.

NIPS-2
Advances in Neural Information Processing Systems 2: Proceedings of the 1989 Conference,
David S. Touretzky, ed., 1990.

NIPS-3
Advances in Neural Information Processing Systems 3: Proceedings of the 1990 Conference,
Richard Lippmann, John E. Moody, and David S. Touretzky, eds., 1991.

NIPS-4
Advances in Neural Information Processing Systems 4: Proceedings of the 1991 Conference,
John E. Moody, Stephen J. Hanson, and Richard P. Lippmann, eds., 1992.

NIPS-5
Advances in Neural Information Processing Systems 5: Proceedings of the 1992 Conference,
Stephen J. Hanson, Jack D. Cowan, and C. Lee Giles, eds., 1993.

NIPS-6
Advances in Neural Information Processing Systems 6: Proceedings of the 1993 Conference,
Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, eds., 1994.

Published by The MIT Press

NIPS-7
Advances in Neural Information Processing Systems 7: Proceedings of the 1994 Conference,
Gerald Tesauro, David S. Touretzky, and Todd K. Leen, eds., 1995.

NIPS-8
Advances in Neural Information Processing Systems 8: Proceedings of the 1995 Conference,
David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, eds., 1996.

NIPS-9
Advances in Neural Information Processing Systems 9: Proceedings of the 1996 Conference,
Michael C. Mozer, Michael I. Jordan, and Thomas Petsche, eds., 1997.

NIPS-10
Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference,
Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, eds., 1998.

NIPS-11
Advances in Neural Information Processing Systems 11: Proceedings of the 1998 Conference,
Michael J. Kearns, Sara A. Solla, and David A. Cohn, eds., 1999.

Advances in Large Margin Classifiers,
Alexander J. Smola, Peter L. Bartlett, Bernhard Schölkopf, and Dale Schuurmans, eds., 2000

Advances in Large Margin Classifiers

edited by
Alexander J. Smola
Peter L. Bartlett
Bernhard Schölkopf
Dale Schuurmans

The MIT Press
Cambridge, Massachusetts
London, England

Printed and bound in the United States of America

Library of Congress Cataloging-in-Publication Data

Advances in large margin classifiers / edited by Alexander J. Smola ... [et al.].
p. cm.
Includes bibliographical references and index.
ISBN 0-262-19448-1 (hc : alk. paper)
1. Machine learning. 2. Algorithms. 3. Kernel functions. I. Smola, Alexander J.

Q325.5.A34 2000
006.3'1--dc21

00-027641

Contents

Series Foreword

The yearly Neural Information Processing Systems (NIPS) workshops bring together scientists with broadly varying backgrounds in statistics, mathematics, computer science, physics, electrical engineering, neuroscience, and cognitive science, unified by a common desire to develop novel computational and statistical strategies for information processing, and to understand the mechanisms for information processing in the brain. As opposed to conferences, these workshops maintain a flexible format that both allows and encourages the presentation and discussion of work in progress, and thus serve as an incubator for the development of important new ideas in this rapidly evolving field.

The Series Editors, in consultation with workshop organizers and members of the NIPS Foundation Board, select specific workshop topics on the basis of scientific excellence, intellectual breadth, and technical impact. Collections of papers chosen and edited by the organizers of specific workshops are built around pedagogical introductory chapters, while research monographs provide comprehensive descriptions of workshop-related topics, to create a series of books that provides a timely, authoritative account of the latest developments in the exciting field of neural computation.

Michael I. Jordan, Sara A. Solla

Preface

The concept of Large Margins has recently been identified as a unifying principle for analyzing many different approaches to the problem of learning to classify data from examples, including Boosting, Mathematical Programming, Neural Networks and Support Vector Machines. The fact that it is the *margin* or confidence level of a classification (i.e., a *scale* parameter) rather than the raw training error that matters has become a key tool in recent years when dealing with classifiers. The present volume shows that this applies both to the theoretical analysis and to the design of algorithms.

Whilst the origin of some of these methods dates back to the work of Vapnik, Mangasarian and others in the 1960s, it took until the 1990s until applications on large real-world problems began. This is due to both the computational resources that recently become available, and theoretical advances, for instance regarding the nonlinear generalization of algorithms. At present, algorithms that explicitly or implicitly exploit the concept of margins are among the most promising approaches to learning from data.

A two-day workshop on this topic was organized at the annual Neural Information Processing Systems (NIPS) conference, held in Breckenridge, Colorado, in December 1998. We are indebted to the people who helped make this happen. In particular we would like to thank the NIPS workshop chairs Rich Zemel and Sue Becker, the conference chair Sara Solla, and all the workshop speakers and attendees who contributed to lively discussions.

The present volume contains a number of papers based on talks presented at the workshop along with a few articles describing results obtained since the workshop has taken place. Although it is far too early to give a final analysis of Large Margin Classifiers, this book attempts to provide a first overview of the subject. We hope that it will help making large margin techniques part of the standard toolbox in data analysis and prediction, and that it will serve as a starting point for further research.

Alexander J. Smola, Peter L. Bartlett, Bernhard Schölkopf, Dale Schuurmans
Canberra, Cambridge, Waterloo, October 1999

1 Introduction to Large Margin Classifiers

The aim of this chapter is to provide a brief introduction to the basic concepts of large margin classifiers for readers unfamiliar with the topic. Moreover it is aimed at establishing a common basis in terms of notation and equations, upon which the subsequent chapters will build (and refer to) when dealing with more advanced issues.

1.1 A Simple Classification Problem

training data

Assume that we are given a set of training data

$$X := \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subseteq \mathbb{R}^N \text{ where } m \in \mathbb{N} \tag{1.1}$$

labels

together with corresponding labels

$$Y := \{y_1, \ldots, y_m\} \subseteq \{-1, 1\}. \tag{1.2}$$

The goal is to find some decision function $g : \mathbb{R}^N \to \{-1, 1\}$ that accurately predicts the labels of unseen data points $(\mathbf{x}, y)$. That is, we seek a function g that minimizes the classification error, which is given by the probability that $g(\mathbf{x}) \neq y$. A common approach to representing decision functions is to use a real valued prediction function $f : \mathbb{R}^N \to \mathbb{R}$ whose output is passed through a sign threshold to yield the final classification $g(\mathbf{x}) = \operatorname{sgn}(f(\mathbf{x}))$. Let us start with a simple example: linear decision functions. In this case the unthresholded prediction is given by a simple linear function of the input vector $\mathbf{x}$

linear decision function

$$g(\mathbf{x}) := \operatorname{sgn}(f(\mathbf{x})) \text{ where } f(\mathbf{x}) = (\mathbf{x} \cdot \mathbf{w}) + b \text{ for } \mathbf{w} \in \mathbb{R}^N \text{ and } b \in \mathbb{R}. \tag{1.3}$$

This gives a classification rule whose decision boundary $\{\mathbf{x} | f(\mathbf{x}) = 0\}$ is an $N - 1$ dimensional hyperplane separating the classes "+1" and "−1" from each other. Figure 1.1 depicts the situation. The problem of learning from data can be formulated as finding a set of parameters $(\mathbf{w}, b)$ such that $\operatorname{sgn}((\mathbf{w} \cdot \mathbf{x}_i) + b) = y_i$ for all $1 \leq i \leq m$. However, such a solution may not always exist, in particular if we are dealing with noisy data. For instance, consider Figure 1.1 with the triangle replaced by an open circle. This raises the question what to do in such a situation.

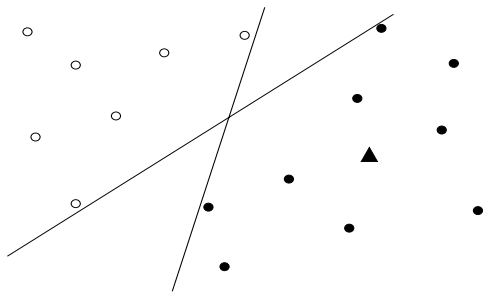


Figure 1.1 A linearly separable classification problem. Note that there may be several possible solutions as depicted by the two lines. The problem becomes non-separable if we replace the triangle by an open circle; in which case no solution $(\mathbf{w}, b)$ exists.

1.1.1 Bayes Optimal Solution

Under the assumption that the data X, Y was generated from a probability distribution $p(\mathbf{x}, y)$ on $\mathbb{R}^N \times \{-1, 1\}$ and that p is known, it is straightforward to find a function that minimizes the probability of misclassification

$$R(g) := \int_{\mathbb{R}^N \times \{-1,1\}} 1_{\{g(\mathbf{x}) \neq y\}} \; p(\mathbf{x}, y) d\mathbf{x} dy. \tag{1.4}$$

Bayes optimal decision function

This function satisfies

$$g(\mathbf{x}) = \text{sgn } (p(\mathbf{x}, 1) - p(\mathbf{x}, -1)) \,. \tag{1.5}$$

Consider a practical example.

Example 1.1 Two Gaussian Clusters

Assume that the two classes "+1" and "−1" are generated by two Gaussian clusters with the same covariance matrix Σ centered at $\boldsymbol{\mu}_+$ and $\boldsymbol{\mu}_-$ respectively

$$p(\mathbf{x}, y) = \frac{1}{2(2\sigma)^{N/2}|\Sigma|^{1/2}} \begin{cases} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_+)^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_+)} & \text{if } y = +1 \\ e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_-)^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_-)} & \text{if } y = -1. \end{cases} \tag{1.6}$$

Since the boundaries completely determine the decision function, we seek the set of points where $p(\mathbf{x}, +1) = p(\mathbf{x}, -1)$. In the case of (1.6) this is equivalent to seeking $\mathbf{x}$ such that

$$(\mathbf{x} - \boldsymbol{\mu}_+)^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_+) \;\; = \;\; (\mathbf{x} - \boldsymbol{\mu}_-)^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}_-). \tag{1.7}$$

By rearranging we find that this condition is equivalent to

$$\begin{aligned} \mathbf{x}^\top \Sigma^{-1} \mathbf{x} - 2\boldsymbol{\mu}_+^\top \Sigma^{-1} \mathbf{x} + \boldsymbol{\mu}_+^\top \Sigma^{-1} \boldsymbol{\mu}_+ - \mathbf{x}^\top \Sigma^{-1} \mathbf{x} + 2\boldsymbol{\mu}_-^\top \Sigma^{-1} \mathbf{x} - \boldsymbol{\mu}_-^\top \Sigma^{-1} \boldsymbol{\mu}_- = 0 \\ 2(\boldsymbol{\mu}_+^\top \Sigma^{-1} - \boldsymbol{\mu}_-^\top \Sigma^{-1}) \mathbf{x} - (\boldsymbol{\mu}_+^\top \Sigma^{-1} \boldsymbol{\mu}_+ - \boldsymbol{\mu}_-^\top \Sigma^{-1} \boldsymbol{\mu}_-) = 0 \end{aligned} \tag{1.8}$$

The latter form is equivalent to having a linear decision function determined by

$$f(\mathbf{x}) = \left((\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)^\top \Sigma^{-1}\right) \mathbf{x} - \tfrac{1}{2}(\boldsymbol{\mu}_+^\top \Sigma^{-1} \boldsymbol{\mu}_+ - \boldsymbol{\mu}_-^\top \Sigma^{-1} \boldsymbol{\mu}_-). \tag{1.9}$$

linear discriminant

Hence in this simple example the Bayes optimal classification rule is linear.

Problems arise, however, if $p(\mathbf{x}, y)$ is not known (as generally happens in practice). In this case one has to obtain a good *estimate* of $g(\mathbf{x}) = \operatorname{sgn}(f(\mathbf{x}))$ from the training data X, Y. A famous example of an algorithm for linear separation is the perceptron algorithm.

1.1.2 The Perceptron Algorithm

The *perceptron algorithm* is "incremental," in the sense that small changes are made to the weight vector in response to each labelled example in turn. For any *learning rate* $\eta > 0$, the algorithm acts sequentially as shown in Table 1.1. Notice

Algorithm 1.1 : Basic Perceptron Algorithm.

argument: `Training sample,` $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathcal{X}$, $Y = \{y_1, \ldots, y_m\} \subset \{\pm 1\}$
`Learning rate,` η
returns: `Weight vector` $\mathbf{w}$ `and threshold` b.
function `Perceptron`(X, Y, η)
  initialize $\mathbf{w}, b = 0$
  repeat
    for all i `from` $i = 1, \ldots, m$
      `Compute` $g(\mathbf{x}_i) = \operatorname{sgn}((\mathbf{w} \cdot \mathbf{x}_i) + b)$
      `Update` $\mathbf{w}, b$ `according to`

$$\begin{aligned} \mathbf{w}' &= \mathbf{w} + (\eta/2)\,(y_i - g(\mathbf{x}_i))\,\mathbf{x}_i \\ b' &= b + (\eta/2)\,(y_i - g(\mathbf{x}_i))\,. \end{aligned}$$

    endfor
  until `for all` $1 \le i \le m$ `we have` $g(\mathbf{x}_i) = y_i$
  return $f : \mathbf{x} \mapsto (\mathbf{w} \cdot \mathbf{x}) + b$
end

that $(\mathbf{w}, b)$ is only updated on a labelled example if the perceptron in state $(\mathbf{w}, b)$ *misclassifies* the example. It is convenient to think of the algorithm as maintaining the hypothesis $g : \mathbf{x} \mapsto \operatorname{sgn}((\mathbf{w} \cdot \mathbf{x}) + b)$, which is updated each time it misclassifies an example. The algorithm operates on a training sample by repeatedly cycling through the m examples, and when it has completed a cycle through the training data without updating its hypothesis, it returns that hypothesis.

perceptron algorithm

The following result shows that if the training sample is consistent with some simple perceptron, then this algorithm converges after a finite number of iterations. In this theorem, $\mathbf{w}^*$ and b^* define a decision boundary that correctly classifies all training points, and every training point is at least distance ρ from the decision boundary.

Theorem 1.1 Convergence of the Perceptron Algorithm
Suppose that there exists a $\rho > 0$, a weight vector $\mathbf{w}^*$ satisfying $\|\mathbf{w}^*\| = 1$, and a threshold b^* such that

$$y_i \left((\mathbf{w}^* \cdot \mathbf{x}_i) + b^*\right) \geq \rho \text{ for all } 1 \leq i \leq m. \tag{1.10}$$

Then for all $\eta > 0$, the hypothesis maintained by the perceptron algorithm converges after no more than $(b^{*2} + 1)(R^2 + 1)/\rho^2$ updates, where $R = \max_i \|x_i\|$. Clearly, the limiting hypothesis is consistent with the training data (X, Y).

Proof [Novikov, 1962] Let $(\mathbf{w}_j, b_j)$ be the state maintained immediately before the jth update occuring at, say, example $(\mathbf{x}_i, y_i)$. To measure the progress of the algorithm, we consider the evolution of the *angle* between $(\mathbf{w}_j, b_j)$ and $(\mathbf{w}^*, b^*)$ and note that the inner product $((\mathbf{w}_j, b_j) \cdot (\mathbf{w}^*, b^*))$ grows steadily with each update. To see this, note that $(\mathbf{w}_j, b_j)$ is only updated when the corresponding hypothesis g_j misclassifies y_i, which implies that $y_i - g_j(\mathbf{x}_i) = 2y_i$. Therefore,

$$\begin{aligned}
((\mathbf{w}_{j+1}, b_{j+1}) \cdot (\mathbf{w}^*, b^*)) &= ([(\mathbf{w}_j, b_j) + (\eta/2)(y_i - g_j(\mathbf{x}_i))(\mathbf{x}_i, 1)] \cdot (\mathbf{w}^*, b^*)) \\
&= ((\mathbf{w}_j, b_j) \cdot (\mathbf{w}^*, b^*)) + \eta y_i ((\mathbf{x}_i, 1) \cdot (\mathbf{w}^*, b^*)) \\
&\geq ((\mathbf{w}_j, b_j) \cdot (\mathbf{w}^*, b^*)) + \eta\rho \\
&\geq j\eta\rho.
\end{aligned}$$

On the other hand, the norm of $(\mathbf{w}_j, b_j)$ cannot grow too fast, because on an update we have $y_i((\mathbf{w}_j \cdot \mathbf{x}_i) + b_j) < 0$, and therefore

$$\begin{aligned}
\|(\mathbf{w}_{j+1}, b_{j+1})\|^2 &= \|(\mathbf{w}_j, b_j) + \eta y_i (\mathbf{x}_i, 1)\|^2 \\
&= \|(\mathbf{w}_j, b_j)\|^2 + 2\eta y_i ((\mathbf{x}_i, 1) \cdot (\mathbf{w}_j, b_j)) + \eta^2 \|(\mathbf{x}_i, 1)\|^2 \\
&\leq \|(\mathbf{w}_j, b_j)\|^2 + \eta^2 \|(\mathbf{x}_i, 1)\|^2 \\
&\leq j\eta^2 (R^2 + 1).
\end{aligned}$$

Combining these two observations with the Cauchy-Schwarz inequality shows that

$$\begin{aligned}
\sqrt{j\eta^2(R^2 + 1)} &\geq \|(\mathbf{w}_{j+1}, b_{j+1})\| \\
&\geq \frac{((\mathbf{w}_{j+1}, b_{j+1}) \cdot (\mathbf{w}^*, b^*))}{\sqrt{1 + b^{*2}}} \\
&\geq \frac{j\eta\rho}{\sqrt{1 + b^{*2}}},
\end{aligned}$$

and thus $j \leq (1 + b^{*2})(R^2 + 1)/\rho^2$ as desired. ■

Since the perceptron algorithm makes an update at least once in every cycle through the training data, and each iteration involves $O(N)$ computation steps, this theorem implies that the perceptron algorithm has time complexity $O((R^2 + 1)mN/\rho^2)$.

1.1.3 Margins

The quantity ρ plays a crucial role in the previous theorem, since it determines how well the two classes can be separated and consequently how fast the perceptron learning algorithm converges. This quantity ρ is what we shall henceforth call a *margin*.

Definition 1.2 Margin and Margin Errors
Denote by $f : \mathbb{R}^N \to \mathbb{R}$ a real valued hypothesis used for classification. Then

$$\rho_f(\mathbf{x}, y) := yf(\mathbf{x}), \tag{1.11}$$

margin

i.e., it is the margin by which the pattern $\mathbf{x}$ is classified correctly (so that a negative value of $\rho_f(\mathbf{x}, y)$ corresponds to an incorrect classification). Moreover denote by

$$\rho_f := \min_{1 \le i \le m} \rho_f(\mathbf{x}_i, y_i) \tag{1.12}$$

minimum margin

the minimum margin over the whole sample. It is determined by the "worst" classification on the whole training set X, Y.

It appears to be desirable to have classifiers that achieve a large margin ρ_f since one might expect that an estimate that is "reliable" on the training set will also perform well on unseen examples. Moreover such an algorithm is more robust with respect to both patterns and parameters:

- Intuitively, for a pattern $\mathbf{x}$ that is far from the decision boundary $\{\mathbf{x}|f(\mathbf{x}) = 0\}$ slight perturbations to $\mathbf{x}$ will not change its classification $\operatorname{sgn}(f(\mathbf{x}))$. To see this, note that if $f(\mathbf{x})$ is a continuous function in $\mathbf{x}$ then small variations in $\mathbf{x}$ will translate into small variations in $f(\mathbf{x})$. Therefore, if $y_i f(\mathbf{x}_i)$ is much larger than zero, $y_i f(\mathbf{x}_i \pm \boldsymbol{\varepsilon})$ will also be positive for small $\boldsymbol{\varepsilon}$. (See, for example, Duda and Hart [1973].)

robustness in patterns

- Similarly, a slight perturbation to the function f will not affect any of the resulting classifications on the training data $(\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$. Assume that $f_{\mathbf{w}}(\mathbf{x})$ is continuous in its parameters $\mathbf{w}$. Then, again, if $y_i f_{\mathbf{w}}(\mathbf{x}_i)$ is much larger than zero, $y_i f_{\mathbf{w}\pm\boldsymbol{\varepsilon}}(\mathbf{x}_i)$ will also be positive for small $\boldsymbol{\varepsilon}$.

robustness in parameters

1.1.4 Maximum Margin Hyperplanes

As pointed out in the previous section, it is desirable to have an estimator with a large margin. This raises the question whether there exists an estimator with *maximum* margin, i.e., whether there exists some f^* with

$$f^* := \operatorname*{argmax}_f \rho_f = \operatorname*{argmax}_f \min_i y_i f(\mathbf{x}_i). \tag{1.13}$$

Without some constraint on the size of $\mathbf{w}$, this maximum does not exist. In Theorem 1.1, we constrained $\mathbf{w}^*$ to have unit length. If we define $f : \mathbb{R}^N \to \mathbb{R}$

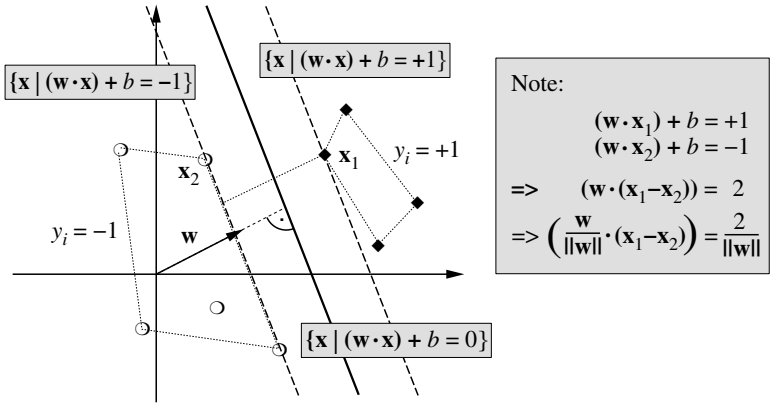


Figure 1.2 A binary classification toy problem: separate balls from diamonds. The *optimal hyperplane* is orthogonal to the shortest line connecting the convex hulls of the two classes (dotted), and intersects it half-way between the two classes. The problem being separable, there exists a weight vector $\mathbf{w}$ and a threshold b such that $y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) > 0$ $(i = 1, \ldots, m)$. Rescaling $\mathbf{w}$ and b such that the point(s) closest to the hyperplane satisfy $|(\mathbf{w} \cdot \mathbf{x}_i) + b| = 1$, we obtain a *canonical* form $(\mathbf{w}, b)$ of the hyperplane, satisfying $y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1$. Note that in this case, the minimum Euclidean distance between the two classes (i.e., twice the margin), measured perpendicularly to the hyperplane, equals $2/\|\mathbf{w}\|$. This can be seen by considering two points $\mathbf{x}_1, \mathbf{x}_2$ on opposite sides of the margin, i.e., $(\mathbf{w} \cdot \mathbf{x}_1) + b = 1, (\mathbf{w} \cdot \mathbf{x}_2) + b = -1$, and projecting them onto the hyperplane normal vector $\mathbf{w}/\|\mathbf{w}\|$.

by

$$f(\mathbf{x}) = \frac{(\mathbf{w} \cdot \mathbf{x}) + b}{\|\mathbf{w}\|}, \tag{1.14}$$

then the maximum margin f is defined by the weight vector and threshold that satisfy

optimal hyperplane

$$\mathbf{w}^*, b^* = \operatorname*{argmax}_{\mathbf{w},b} \min_{i=1}^{m} \frac{y_i((\mathbf{w} \cdot \mathbf{x}_i) + b)}{\|\mathbf{w}\|} \tag{1.15}$$

$$= \operatorname*{argmax}_{\mathbf{w},b} \min_{i=1}^{m} y_i \ \operatorname{sgn}((\mathbf{w} \cdot \mathbf{x}_i) + b) \left\| \frac{(\mathbf{w} \cdot \mathbf{x}_i)}{\|\mathbf{w}\|^2}\mathbf{w} + \frac{b}{\|\mathbf{w}\|^2}\mathbf{w} \right\| \tag{1.16}$$

Euclidean Margin

The formulation (1.16) has a simple geometric interpretation: $-b\mathbf{w}/\|\mathbf{w}\|^2$ is the vector in direction $\mathbf{w}$ that ends right on the decision hyperplane (since $\left(\mathbf{w} \cdot (-b\mathbf{w}/\|\mathbf{w}\|^2)\right) = -b$), and for a vector $\mathbf{x}_i$, $(\mathbf{w} \cdot \mathbf{x}_i)\mathbf{w}/\|\mathbf{w}\|^2$ is the projection of $\mathbf{x}_i$ onto $\mathbf{w}$. Therefore, we are interested in maximizing the length of the vector differences $(\mathbf{w} \cdot \mathbf{x}_i)\mathbf{w}/\|\mathbf{w}\|^2 - (-b\mathbf{w}/\|\mathbf{w}\|^2)$ appropriatedly signed by $y_i g(\mathbf{x}_i)$.

optimization problems

The maxi-min problem (1.15) can be easily transformed into an equivalent constrained optimization task by conjecturing a lower bound on the margin, ρ,

and maximizing ρ subject to the constraint that it really is a lower bound:

$$\begin{aligned}
&\mathbf{w}^*, b^*, \rho^* \\
&= \underset{\mathbf{w},b,\rho}{\operatorname{argmax}} \quad \rho \quad \text{subject to } \frac{y_i((\mathbf{w}\cdot\mathbf{x}_i)+b)}{\|\mathbf{w}\|} \geq \rho \text{ for } 1 \leq i \leq m && (1.17)\\
&= \underset{\mathbf{w},b,\rho}{\operatorname{argmax}} \quad \rho \quad \text{subject to } \|\mathbf{w}\| = 1 \text{ and } y_i((\mathbf{w}\cdot\mathbf{x}_i)+b) \geq \rho \text{ for } 1 \leq i \leq m && (1.18)\\
&= \underset{\mathbf{w},b}{\operatorname{argmin}} \quad \|\mathbf{w}\|^2 \quad \text{subject to } y_i((\mathbf{w}\cdot\mathbf{x}_i)+b) \geq 1 \text{ for } 1 \leq i \leq m && (1.19)
\end{aligned}$$

quadratic program

This last formulation is in the form of a quadratic programming problem, which can be easily handled using standard optimizers [Luenberger, 1973, Bertsekas, 1995].

Notice that (1.18) is in a particularly intuitive form. This formulation states that we are seeking a weight vector $\mathbf{w}$ that obtains large dot products $y_i(\mathbf{w}\cdot\mathbf{x}_i)$, but constrain the weight vector to lie on the unit sphere to prevent obtaining such large dot products "for free" by scaling up $\mathbf{w}$. Interesting variants of problem (1.18) are obtained by choosing different norms to constrain the length of the weight vector. For example, constraining $\mathbf{w}$ to lie on the unit ℓ_1 sphere instead of the unit ℓ_2 sphere gives the problem of determining

$$\begin{aligned}
&\mathbf{w}^*, b^*, \rho^* \\
&= \underset{\mathbf{w},b,\rho}{\operatorname{argmax}} \quad \rho \quad \text{subject to } \|\mathbf{w}\|_1 = 1 \text{ and } y_i((\mathbf{w}\cdot\mathbf{x}_i)+b) \geq \rho \text{ for } 1 \leq i \leq m && (1.20)
\end{aligned}$$

ℓ_∞ margin

which can easily be shown to be in the form of a linear programming problem. Mangasarian [1997] shows that this is equivalent to finding the weight vector and threshold that maximize the minimum ℓ_∞ distance between the training patterns and the decision hyperplane, in a direct analogue to the original Euclidean formulation (1.15).

Similarly, the constraint that $\mathbf{w}$ lie on the unit ℓ_∞ sphere yields the problem

$$\begin{aligned}
&\mathbf{w}^*, b^*, \rho^* \\
&= \underset{\mathbf{w},b,\rho}{\operatorname{argmax}} \quad \rho \quad \text{subject to } \|\mathbf{w}\|_\infty = 1 \text{ and } y_i((\mathbf{w}\cdot\mathbf{x}_i)+b) \geq \rho \text{ for } 1 \leq i \leq m && (1.21)
\end{aligned}$$

ℓ_1 margin

which is also a linear programming problem, but now equivalent to finding the weight vector and threshold that maximize the minimum ℓ_1 distance between the training patterns and the decision hyperplane. In general, constraining $\mathbf{w}$ to lie on the unit ℓ_p sphere yields a convex programming problem

$$\begin{aligned}
&\mathbf{w}^*, b^*, \rho^* \\
&= \underset{\mathbf{w},b,\rho}{\operatorname{argmax}} \quad \rho \quad \text{subject to } \|\mathbf{w}\|_p = 1 \text{ and } y_i((\mathbf{w}\cdot\mathbf{x}_i)+b) \geq \rho \text{ for } 1 \leq i \leq m && (1.22)
\end{aligned}$$

ℓ_q margin

which is equivalent to finding the weight vector and threshold that maximize the minimum ℓ_q distance between the training patterns and the decision hyperplane, where ℓ_p and ℓ_q are conjugate norms, i.e., such that $\frac{1}{p} + \frac{1}{q} = 1$ [Mangasarian, 1997].

In solving any of these constrained optimization problems, there is a notion of *critical constraints*, i.e., those inequality constraints that are satisfied as equalities

by the optimal solution. In our setting, constraints correspond to training examples $(\mathbf{x}_i, y_i)$, $1 \leq i \leq m$, and the *critical* constraints are given by those training examples that lie right on the margin a distance ρ from the optimal hyperplane (cf. Figure 1.2). These critical training patterns are called *Support Vectors*.

Support Vectors

Notice that all the remaining examples of the training set are irrelevant: for non-critical examples the corresponding constraint $y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1$ in (1.19) does not play a role in the optimization, and therefore these points could be removed from the training set without affecting the results. This nicely captures our intuition of the problem: the hyperplane (cf. Figure 1.2) is completely determined by the patterns closest to it, the solution should not depend on the other examples.

In practice, a separating hyperplane may not exist, e.g., if a high noise level causes a large overlap of the classes. The previous maximum margin algorithms perform poorly in this case because the maximum achievable minimum margin is negative, and this means the critical constraints are the mislabelled patterns that are furthest from the decision hyperplane. That is, the solution hyperplane is determined entirely by misclassified examples! To overcome the sensitivity to noisy training patterns, a standard approach is to allow for the possibility of examples violating the constraint in (1.19) by introducing *slack variables* [Bennett and Mangasarian, 1992, Cortes and Vapnik, 1995, Vapnik, 1995]

soft margin hyperplane

$$\xi_i \geq 0, \text{ for all } i = 1, \ldots, m, \tag{1.23}$$

slack variables

along with relaxed constraints

$$y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1 - \xi_i, \text{ for all } i = 1, \ldots, m. \tag{1.24}$$

A classifier which generalizes well is then found by controlling both the size of $\mathbf{w}$ and the number of training errors, minimizing the objective function

$$\tau(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m} \xi_i \tag{1.25}$$

subject to the constraints (1.23) and (1.24), for some value of the constant $C > 0$.

In the following section, we shall see why the size of $\mathbf{w}$ is a good measure of the complexity of the classifier.

1.2 Theory

In order to provide a theoretical analysis of the learning problem we have to introduce a few definitions and assumptions about the process generating the data.

1.2.1 Basic Assumptions

independently identically distributed

We assume that the training data X, Y is drawn independently and identically distributed (iid) according to some probability measure $p(\mathbf{x}, y)$. This means that

all examples $(\mathbf{x}_i, y_i)$ are drawn from $p(\mathbf{x}, y)$ regardless of the other examples or the index i.

This assumption is stronger than it may appear at first glance. For instance, time series data fails to satisfy the condition, since the observations are typically dependent, and their statistics might depend on the index i.

In (1.4), we defined the functional $R(g)$ of a decision function g as the probability of misclassification. We can generalize this definition to apply to prediction functions f as well as thresholded decision functions g. This yields what we call the risk functional.

Definition 1.3 Risk Functional

Denote by $c(\mathbf{x}, y, f(\mathbf{x})) : \mathbb{R}^N \times \mathbb{R} \times \mathbb{R} \to [0, \infty)$ a cost function and by $p(\mathbf{x}, y)$ a probability measure as described above. Then the risk functional for a function $f : \mathbb{R}^N \to \mathbb{R}$ is defined as

Expected Risk

$$R(f) := \int_{\mathbb{R}^N \times \mathbb{R}} c(\mathbf{x}, y, f(\mathbf{x}))\, dp(\mathbf{x}, y). \tag{1.26}$$

Moreover the *empirical* risk functional for an m–sample X, Y is given by

$$R_{\text{emp}}(f) := \tfrac{1}{m} \sum_{i=1}^{m} c(\mathbf{x}_i, y_i, f(\mathbf{x}_i)). \tag{1.27}$$

Empirical Risk

For thresholded decision functions $g : \mathbb{R}^N \to \{-1, 1\}$ we often use 0–1 classification error as the cost function $c(\mathbf{x}, y, g(\mathbf{x})) = 1_{\{g(\mathbf{x}) \neq y\}}$. In this case we obtain the risk functional defined in (1.4) (the probability of misclassification),

$$R(g) := \Pr\{g(\mathbf{x}) \neq y\}. \tag{1.28}$$

In this case, the empirical risk functional is

$$R_{\text{emp}}(g) := \tfrac{1}{m} \sum_{i=1}^{m} 1_{\{g(\mathbf{x}_i) \neq y_i\}}, \tag{1.29}$$

which is just the training error.

margin error

Finally we need a quantity called the *margin error*, which is given by the proportion of training points that have margin less than ρ, i.e.,

$$R_\rho(f) := \tfrac{1}{m} \sum_{i=1}^{m} 1_{\{y_i f(\mathbf{x}_i) < \rho\}}. \tag{1.30}$$

This empirical estimate of risk counts a point as an error if it is either incorrectly classified or correctly classified by with margin less than ρ.

While one wants to minimize the risk $R(g)$ this is hardly ever possible since $p(\mathbf{x}, y)$ is unknown. Hence one may only resort to minimizing $R_{\text{emp}}(g)$ which is based on the training data. This, however, is not an effective method by itself—just consider an estimator that memorizes all the training data X, Y and generates random outputs for any other data. This clearly would have an empirical risk $R_{\text{emp}}(g) = 0$ but would obtain a true risk $R(g) = 0.5$ (assuming the finite training sample has measure 0).

The solution is to take the complexity of the estimate g into account as well, which will be discussed in the following sections.

1.2.2 Error Bounds for Thresholded Decision Functions

The central result of this analysis is to relate the number of training examples, the training set error, and the complexity of the hypothesis space to the generalization error. For thresholded decision functions, an appropriate measure for the complexity of the hypothesis space is the Vapnik-Chervonenkis (VC) dimension.

VC dimension

Definition 1.4 VC dimension (Vapnik and Chervonenkis, 1971)
The VC dimension h of a space of $\{-1,1\}$-valued functions, G, is the size of the largest subset of domain points that can be labelled arbitrarily by choosing functions only from G.

The VC dimension can be used to prove high probability bounds on the error of a hypothesis chosen from a class of decision functions G—this is the famous result of Vapnik and Chervonenkis [1971]. The bounds have since been improved slightly by Talagrand [1994]—see also [Alexander, 1984].

Theorem 1.5 VC Upper Bound
Let G be a class of decision functions mapping $\mathbb{R}^N$ to $\{-1,1\}$ that has VC dimension h. For any probability distribution $p(\mathbf{x},y)$ on $\mathbb{R}^N \times \{-1,1\}$, with probability at least $1-\delta$ over m random examples $\mathbf{x}$, for any hypothesis g in G the risk functional with 0–1 loss is bounded by

$$R(g) \leq R_{\text{emp}}(g) + \sqrt{\frac{c}{m}\left(h + \ln\left(\frac{1}{\delta}\right)\right)} \tag{1.31}$$

where c is a universal constant. Furthermore, if $g^* \in G$ minimizes $R_{\text{emp}}(\cdot)$, then with probability $1-\delta$

$$R(g^*) \leq \inf_{g \in G} R(g) + \sqrt{\frac{c}{m}\left(h + \ln\left(\frac{1}{\delta}\right)\right)} \tag{1.32}$$

(A short proof of this result is given by Long [1998], but with worse constants than Talagrand's.) These upper bounds are asymptotically close to the best possible, since there is also a lower bound with the same form:

Theorem 1.6 VC Lower Bound
Let G be a hypothesis space with finite VC dimension $h \geq 1$. Then for any learning algorithm there exist distributions such that with probability at least δ over m random examples, the error of its hypothesis g satisfies

$$R(g) \geq \inf_{g' \in G} R(g') + \sqrt{\frac{c}{m}\left(h + \ln\left(\frac{1}{\delta}\right)\right)} \tag{1.33}$$

where c is a universal constant.

(Results of this form have been given by Devroye and Lugosi [1995], Simon [1996], Anthony and Bartlett [1999], using ideas from Ehrenfeucht et al. [1989].)

Theorems 1.5 and 1.6 give a fairly complete characterization of the generalization error that can be achieved by choosing decision functions from a class G. However, this characterization suffers from two drawbacks.

- The first drawback is that the VC dimension must actually be determined (or at least bounded) for the class of interest—and this is often not easy to do. (However, bounds on the VC dimension h have been computed for many natural decision function classes, including parametric classes involving standard arithmetic and boolean operations. See Anthony and Bartlett [1999] for a review of these results.)
- The second (more serious) drawback is that the analysis ignores the *structure* of the mapping from training samples to hypotheses, and concentrates solely on the *range* of the learner's possible outputs. Ignoring the details of the learning map can omit many of the factors that are *crucial* for determining the success of the learning algorithm in real situations.
For example, consider learning algorithms that operate by first computing a real valued prediction function f from some class F and then thresholding this hypothesis to obtain the final decision function $g(\mathbf{x}) = \operatorname{sgn}(f(\mathbf{x}))$. Here, the VC dimension is a particularly weak method for measuring the representational capacity of the resulting function class $G = \operatorname{sgn}(F)$.
One reason is that the VC dimension of G is not sensitive to the *scale* of F at the accuracy level of interest. That is, it does not pay attention to whether the complexity of the hypothesis class is at a scale that is relevant for the outcome of the predictions.

The first step towards a more refined analysis that takes scale into account is given by Vapnik [1979]. Consider a set $X_0 \subset \mathbb{R}^N$ of input points with norm bounded by $R > 0$ (that is, $\|\mathbf{x}_i\| \leq R$ for $\mathbf{x} \in X_0$), and the set F of bounded linear functions defined on X_0,

$$F = \{\mathbf{x} \mapsto (\mathbf{w} \cdot \mathbf{x}) \;\; | \;\; \|\mathbf{w}\| \leq 1, \, \mathbf{x} \in X_0 \} \tag{1.34}$$

satisfying $|f(\mathbf{x})| \geq \rho$ for all patterns $\mathbf{x}$ in X_0. Then if we consider the set G of linear decision functions obtained by thresholding functions in F, Vapnik [1979] shows

$$\operatorname{VCdim}(G) \leq \min\{R^2/\rho^2, N\} + 1. \tag{1.35}$$

Note that this can be much smaller than the VC dimension of $\operatorname{sgn}(F)$ obtained without taking ρ into account, which is $N + 1$ in this case. Therefore, one could hope to obtain significant benefits by using scale sensitive bounds which give much tighter results for large margin classifiers. Unfortunately, the bound (1.35) does not yet suffice for our purposes, because note that it requires that *all* points (including the test points) satisfy the margin condition, and therefore Theorem 1.5 does not apply in this case. Rigorously obtaining these scale sensitive improvements is the topic we now address. In the following section, we consider scale-sensitive versions

of the VC dimension, and obtain upper and lower bounds on risk in terms of these dimensions.

1.2.3 Margin Dependent Error Bounds for Real Valued Predictors

Definition 1.7 Fat Shattering Dimension
Let F be a set of real valued functions. We say that a set of points $S \subset \mathcal{X}$, which we will index as a vector $\mathbf{x} \in \mathcal{X}^{|S|}$, is *$\rho$-shattered by F* if there is a vector of real numbers $\mathbf{b} \in \mathbb{R}^{|S|}$ such that for any choice of signs $\mathbf{y} \in \{-1, 1\}^{|S|}$ there is a function f in F that satisfies

$$y_i(f(x_i) - b_i) \geq \rho \text{ for } 1 \leq i \leq |S|. \tag{1.36}$$

(That is, $f(x_i) \geq b_i + \rho$ if $y_i = 1$, and $f(x_i) \leq b_i - \rho$ if $y_i = -1$, for all x_i in S. Notice how similar this is to the notion of a minimum margin defined by (1.12).) The *fat shattering dimension* $\text{fat}_F(\rho)$ of the set F is a function from the positive real numbers to the integers which maps a value ρ to the size of the largest ρ-shattered set, if this is finite, or infinity otherwise.

fat shattering

We may think of the fat-shattering dimension of a set of real-valued functions as the VC dimension obtained by thresholding but requiring that outputs are ρ above the threshold for positive classification and ρ below for negative.

The fat-shattering dimension is closely related to a more basic quantity, the covering number of a class of functions.

Definition 1.8 Covering Numbers of a Set
Denote by (S, d) a pseudometric space, $B_r(\mathbf{x})$ the closed ball in S centred at $\mathbf{x}$ with radius r, T a subset of S, and ε some positive constant. Then the covering number $\mathcal{N}(\varepsilon, T)$ is defined as the minimum cardinality (that is, number of elements) of a set of points $T' \subset S$ such that

covering number

$$T \subseteq \bigcup_{\mathbf{x}_i \in T'} B_\varepsilon(\mathbf{x}_i), \tag{1.37}$$

i.e., such that the maximum difference of any element in T and the closest element in T' is less than or equal to ε.

Covering a class of functions F with an ε-cover means that one is able to approximately represent F (which may be of infinite cardinality) by a finite set. For learning, it turns out that it suffices to approximate the restrictions of functions in a class F to finite samples. For a subset X of some domain $\mathcal{X}$, define the pseudometric $\ell_{\infty,X}$ by

$$\ell_{\infty,X}(f, f') = \max_{\mathbf{x} \in X} |f(\mathbf{x}) - f'(\mathbf{x})| \tag{1.38}$$

where f and f' are real-valued functions defined on $\mathcal{X}$. Let $\mathcal{N}(\varepsilon, F, m)$ denote the maximum, over all $X \subset \mathcal{X}$ of size $|X| = m$, of the covering number $\mathcal{N}(\varepsilon, F)$ with respect to $\ell_{\infty,X}$. The following theorem shows that the fat-shattering dimension

is intimately related to these covering numbers. (The upper bound is due to Alon et al. [1997], and the lower bound to Bartlett et al. [1997].)

Theorem 1.9 Bounds on $\mathcal{N}$ in terms of fat_F
Let F be a set of real functions from a domain $\mathcal{X}$ to the bounded interval $[0, B]$. Let $\varepsilon > 0$ and let $m \geq \mathrm{fat}_F(\varepsilon/4)$. Then

$$\frac{\log_2 e}{8}\mathrm{fat}_F(16\varepsilon) \;\leq\; \log_2 \mathcal{N}(\varepsilon, F, m) \;\leq\; 3\,\mathrm{fat}_F(\tfrac{\varepsilon}{4})\log_2^2\left(\frac{4eBm}{\varepsilon}\right). \tag{1.39}$$

Unfortunately, directly bounding $\mathcal{N}$ can be quite difficult in general. Useful tools from functional analysis (which deal with the functional inverse of $\mathcal{N}$ wrt. ϵ, the so called entropy number) for obtaining these bounds have been developed for classes of functions F defined by linear mappings from Hilbert spaces [Carl and Stephani, 1990], and linear functions over kernel expansions [Williamson et al., 1998].

The following result shows that we can use covering numbers to obtain upper bounds on risk in terms of margin error [Shawe-Taylor et al., 1998, Bartlett, 1998].

Theorem 1.10 Bounds on $R(f)$ in terms of $\mathcal{N}$ and ρ
Suppose that F is a set of real-valued functions defined on $\mathcal{X}$, $\varepsilon \in (0, 1)$ and $\rho > 0$. Fix a probability distribution on $\mathcal{X} \times \{-1, 1\}$ and a sample size m. Then the probability that some f in F has $R_\rho(f) = 0$ but $R(f) \geq \varepsilon$ is no more than

$$2\,\mathcal{N}\left(\tfrac{\rho}{2}, F, 2m\right)\, 2^{-\varepsilon m/2}. \tag{1.40}$$

Furthermore,

$$\Pr\left(\text{“some } f \text{ in } F \text{ has } R(f) \geq R_\rho(f) + \varepsilon\text{”}\right) \;\leq\; 2\,\mathcal{N}\left(\tfrac{\rho}{2}, F, 2m\right)\, e^{-\varepsilon^2 m/8}. \tag{1.41}$$

In fact, it is possible to obtain a similar result that depends only on the behaviour of functions in F near the threshold (see [Anthony and Bartlett, 1999] for details).

anatomy of a uniform convergence bound

Let us have a close look at the bound (1.41) on the probability of excessive error. The factor $e^{-\varepsilon^2 m/8}$ in (1.41) stems from a bound of Hoeffding [1963] on the probability of a large deviation of a sum of random variables from its mean. The factor $\mathcal{N}\left(\frac{\rho}{2}, F, 2m\right)$ stems from the fact that the continuous class of functions F was approximated (to accuracy $\rho/2$) by a finite number of functions. The $2m$ is due to the use of a symmetrization argument which is needed to make the overall argument work. Theorem 1.9 shows that this term is bounded by an exponential function of the fat-shattering dimension at scale $\rho/8$.

Interestingly, a similar result holds in regression. (For a review of these uniform convergence results, see [Anthony and Bartlett, 1999]).

Theorem 1.11 Bounds on $R(f)$ for Regression
Suppose that F is a set of functions defined on a domain $\mathcal{X}$ and mapping into the real interval $[0, 1]$. Let p be any probability distribution on $\mathcal{X} \times [0, 1]$, ε any real number between 0 and 1, and $m \in \mathbb{N}$. Then for the quadratic cost function

$c(\mathbf{x}, y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$ we have

$$\Pr\left(\sup_{f\in F}|R(f) - R_{\rm emp}(f)| \geq \varepsilon\right) \;\leq\; 4\,\mathcal{N}\left(\tfrac{\varepsilon}{16}, F, 2m\right)\, e^{-\varepsilon^2 m/32}. \tag{1.42}$$

Comparing with (1.41), notice that the scale of the covering number depends on the desired accuracy ε, whereas in (1.41) it depends on the scale ρ at which the margins are examined.

1.2.4 Error Bounds for Linear Decision Functions

The following result, due to Bartlett and Shawe-Taylor [1999], gives a bound on the fat-shattering dimension of large margin linear classifiers. It has a similar form to the bound (1.35) on the VC dimension of linear functions restricted to certain sets. It improves on a straightforward corollary of that result, and on a result of Gurvits [1997].

Theorem 1.12 Fat Shattering Dimension for Linear Classifiers
Suppose that B_R is the ℓ_2 ball of radius R in $\mathbb{R}^n$, centered at the origin, and consider the set

$$F := \{f_{\mathbf{w}} \;\mid\; f_{\mathbf{w}}(\mathbf{x}) = (\mathbf{w}\cdot\mathbf{x}) \text{ with } \|\mathbf{w}\| \leq 1, \mathbf{x} \in B_R\}. \tag{1.43}$$

Then

$$\mathrm{fat}_F(\rho) \leq \left(\tfrac{R}{\rho}\right)^2. \tag{1.44}$$

Using this result together with Theorems 1.9 and 1.10 gives the following theorem.

Theorem 1.13 Error Bounds for Linear Classifiers
Define the class F of real-valued functions on the ball of radius R as in (1.43). There is a constant c such that, for all probability distributions, with probability at least $1-\delta$ over m independently generated training examples, every $\rho > 0$ and every function $f \in F$ with margin at least ρ on all training examples (i.e., $R_\rho(f) = 0$) satisfies

$$R(f) \;\leq\; \frac{c}{m}\left(\frac{R^2}{\rho^2}\log^2\left(\frac{m}{\rho}\right) + \log\left(\frac{1}{\delta}\right)\right). \tag{1.45}$$

Furthermore, with probability at least $1-\delta$, for all $\rho > 0$, every function f in F has error

$$R(f) \;\leq\; R_\rho(f) + \sqrt{\frac{c}{m}\left(\frac{R^2}{\rho^2}\log^2\left(\frac{m}{\rho}\right) + \log\left(\frac{1}{\delta}\right)\right)}. \tag{1.46}$$

For estimators using a linear programming approach as in [Mangasarian, 1968] one may state the following result which is an improvement, by a log factor, of Theorem 17 in [Bartlett, 1998]. Applying Theorem 1.9, this can be transformed into a generalization bound as well.

Theorem 1.14 Capacity Bounds for Linear Classifiers
There is a constant c such that for the class

$$F_R = \left\{ \mathbf{x} \mapsto \mathbf{w}^T \mathbf{x} \;\; | \;\; \|\mathbf{x}\|_\infty \leq 1, \|\mathbf{w}\|_1 \leq R \right\} \tag{1.47}$$

we have

$$\mathrm{fat}_{F_R}(\varepsilon) \;\; \leq \;\; c \left(\frac{R}{\varepsilon} \right)^2 \ln(2N+2). \tag{1.48}$$

Finally, we can obtain bounds for convex combinations of arbitrary hypotheses from a class G of $\{-1,1\}$-valued functions,

$$\mathrm{co}\,(G) = \left\{ \sum_i \alpha_i g_i \;\; \Bigg| \;\; \alpha_i > 0, \sum_i \alpha_i = 1, g_i \in G \right\}. \tag{1.49}$$

See [Schapire et al., 1998]. These bounds are useful in analysing boosting algorithms; see Section 1.4.

Theorem 1.15 Bounds for Convex Combinations of Hypotheses
Let $p(\mathbf{x}, y)$ be a distribution over $\mathcal{X} \times \{-1, 1\}$, and let X be a sample of m examples chosen iid according to p. Suppose the base-hypothesis space G has VC dimension h, and let $\delta > 0$. Then with probability at least $1 - \delta$ over the random choice of the training set X, Y, every convex combination of functions $f \in \mathrm{co}\,(G)$ satisfies the following bound for all $\rho > 0$.

$$R(f) \leq R_\rho(f) + \sqrt{\frac{c}{m} \left(\frac{h \log^2(m/h)}{\rho^2} + \log\left(\frac{1}{\delta}\right) \right)} \tag{1.50}$$

1.3 Support Vector Machines

We now turn to one of the types of learning algorithms that the present book deals with. For further details, cf. [Vapnik, 1995, 1998, Burges, 1998, Smola and Schölkopf, 1998, Cristianini and Shawe-Taylor, 2000, Schölkopf and Smola, 2000] or the collection [Schölkopf et al., 1999a], which also formed the basis for some of the material presented below.

1.3.1 Optimization Problem

To construct the *Optimal Hyperplane* (cf. Figure 1.2), one solves the following optimization problem:

$$\text{minimize} \qquad \tau(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 \tag{1.51}$$

$$\text{subject to} \qquad y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) \geq 1, \;\; \text{for all } i = 1, \ldots, m. \tag{1.52}$$

This constrained optimization problem is dealt with by introducing Lagrange multipliers $\alpha_i \geq 0$ and a Lagrangian

Lagrangian

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left(y_i((\mathbf{x}_i \cdot \mathbf{w}) + b) - 1\right). \tag{1.53}$$

The Lagrangian L has to be minimized with respect to the *primal variables* $\mathbf{w}$ and b and maximized with respect to the *dual variables* α_i (i.e., a saddle point has to be found). Let us try to get some intuition for this. If a constraint (1.52) is violated, then $y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) - 1 < 0$, in which case L can be increased by increasing the corresponding α_i. At the same time, $\mathbf{w}$ and b will have to change such that L decreases. To prevent $-\alpha_i \left(y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) - 1\right)$ from becoming arbitrarily large, the change in $\mathbf{w}$ and b will ensure that, provided the problem is separable, the constraint will eventually be satisfied.

KKT conditions

Similarly, one can understand that for all constraints which are not precisely met as equalities, i.e., for which $y_i((\mathbf{w} \cdot \mathbf{x}_i) + b) - 1 > 0$, the corresponding α_i must be 0: this is the value of α_i that maximizes L. The latter is the statement of the Karush-Kuhn-Tucker complementarity conditions of optimization theory [Karush, 1939, Kuhn and Tucker, 1951, Bertsekas, 1995].

The condition that at the saddle point, the derivatives of L with respect to the primal variables must vanish,

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0 \text{ and } \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \tag{1.54}$$

leads to

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{1.55}$$

and

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i. \tag{1.56}$$

support vector expansion

The solution vector thus has an expansion in terms of a subset of the training patterns, namely those patterns whose Lagrange multiplier α_i is non-zero. By the Karush-Kuhn-Tucker complementarity conditions these training patterns are the ones for which

$$\alpha_i(y_i((\mathbf{x}_i \cdot \mathbf{w}) + b) - 1) = 0, \quad i = 1, \ldots, m, \tag{1.57}$$

and therefore they correspond precisely to the *Support Vectors* (i.e., critical constraints) discussed in Section 1.1.4. Thus we have the satisfying result that the Support Vectors are the only training patterns that determine the optimal decision hyperplane; all other training patterns are irrelevant and do not appear in the expansion (1.56).

dual optimization problem

By substituting (1.55) and (1.56) into L, one eliminates the primal variables and arrives at the Wolfe dual of the optimization problem [e.g. Bertsekas, 1995]: find

multipliers α_i which

$$\text{maximize} \qquad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j\right) \tag{1.58}$$

$$\text{subject to} \qquad \alpha_i \geq 0 \text{ for all } i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{1.59}$$

The hyperplane decision function can thus be written as

$$g(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \left(\mathbf{x} \cdot \mathbf{x}_i\right) + b\right) \tag{1.60}$$

where b is computed using (1.57).

The structure of the optimization problem closely resembles those that typically arise in Lagrange's formulation of mechanics [e.g. Goldstein, 1986]. In that case also, it is often only a subset of the constraints that are active. For instance, if we keep a ball in a box, then it will typically roll into one of the corners. The constraints corresponding to the walls which are not touched by the ball are irrelevant, the walls could just as well be removed.

Seen in this light, it is not too surprising that it is possible to give a mechanical interpretation of optimal margin hyperplanes [Burges and Schölkopf, 1997]: If we assume that each support vector $\mathbf{x}_i$ exerts a perpendicular force of size α_i and sign y_i on a solid plane sheet lying along the hyperplane, then the solution satisfies the requirements of mechanical stability. The constraint (1.55) states that the forces on the sheet sum to zero; and (1.56) implies that the torques also sum to zero, via $\sum_i \mathbf{x}_i \times y_i \alpha_i \mathbf{w}/\|\mathbf{w}\| = \mathbf{w} \times \mathbf{w}/\|\mathbf{w}\| = 0$.

1.3.2 Feature Spaces and Kernels

To construct *Support Vector Machines*, the optimal hyperplane algorithm is augmented by a method for computing dot products in feature spaces that are *nonlinearly* related to input space [Aizerman et al., 1964, Boser et al., 1992]. The basic idea is to map the data into some other dot product space (called the *feature space*) $\mathcal{F}$ via a nonlinear map

feature space

$$\Phi : \mathbb{R}^N \to \mathcal{F}, \tag{1.61}$$

and then in the space $\mathcal{F}$ perform the linear algorithm described above.

For instance, suppose we are given patterns $\mathbf{x} \in \mathbb{R}^N$ where most information is contained in the d-th order products (monomials) of entries x_j of $\mathbf{x}$, i.e., $x_{j_1} x_{j_2} \cdots x_{j_d}$, where $j_1, \ldots, j_d \in \{1, \ldots, N\}$. There, we might prefer to extract the monomial features first, and work in the feature space $\mathcal{F}$ of all products of d entries.

This approach, however, fails for realistically sized problems: for N-dimensional input patterns, there exist $(N + d - 1)!/(d!(N - 1)!)$ different monomials. Already 16×16 pixel input images (e.g., in character recognition) and a monomial degree $d = 5$ yield a dimensionality of 10^{10}.

This problem can be overcome by noticing that both the construction of the optimal hyperplane in $\mathcal{F}$ (cf. (1.58)) and the evaluation of the corresponding decision function (1.60) only require the evaluation of dot products $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}'))$, and never require the mapped patterns $\Phi(\mathbf{x})$ in explicit form. This is crucial, since in some cases, the dot products can be evaluated by a simple kernel [Aizerman et al., 1964, Boser et al., 1992].

Mercer kernel

$$k(\mathbf{x}, \mathbf{x}') = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')). \tag{1.62}$$

polynomial kernel

For instance, the polynomial kernel

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^d \tag{1.63}$$

can be shown to correspond to a map Φ into the space spanned by all products of exactly d dimensions of $\mathbb{R}^N$ (Poggio [1975], Boser et al. [1992]). For a proof, see Schölkopf [1997]. For $d = 2$ and $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^2$, for example, we have [Vapnik, 1995]

$$(\mathbf{x} \cdot \mathbf{x}')^2 = (x_1^2, x_2^2, \sqrt{2}\, x_1 x_2)(y_1^2, y_2^2, \sqrt{2}\, y_1 y_2)^\top = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')), \tag{1.64}$$

defining $\Phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}\, x_1 x_2)$.

By using $k(\mathbf{x}, \mathbf{x}') = ((\mathbf{x} \cdot \mathbf{x}') + c)^d$ with $c > 0$, we can take into account all product of order up to d (i.e., including those of order smaller than d).

More generally, the following theorem of functional analysis shows that kernels k of positive integral operators give rise to maps Φ such that (1.62) holds [Mercer, 1909, Aizerman et al., 1964, Boser et al., 1992, Dunford and Schwartz, 1963]:

Theorem 1.16 Mercer

positive integral operator

If k is a continuous symmetric kernel of a positive integral operator T, i.e.,

$$(Tf)(\mathbf{x}') = \int_{\mathcal{X}} k(\mathbf{x}, \mathbf{x}') f(\mathbf{x})\, d\mathbf{x} \tag{1.65}$$

with

$$\int_{\mathcal{X} \times \mathcal{X}} k(\mathbf{x}, \mathbf{x}') f(\mathbf{x}) f(\mathbf{x}')\, d\mathbf{x}\, d\mathbf{x}' \geq 0 \tag{1.66}$$

for all $f \in L_2(\mathcal{X})$ ($\mathcal{X}$ being a compact subset of $\mathbb{R}^N$), it can be expanded in a uniformly convergent series (on $\mathcal{X} \times \mathcal{X}$) in terms of T's eigenfunctions ψ_j and positive eigenvalues λ_j,

$$k(\mathbf{x}, \mathbf{x}') = \sum_{j=1}^{N_{\mathcal{F}}} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{x}'), \tag{1.67}$$

where $N_{\mathcal{F}} \leq \infty$ is the number of positive eigenvalues.

An equivalent way to characterize Mercer kernels is that they give rise to positive matrices $K_{ij} := k(\mathbf{x}_i, \mathbf{x}_j)$ for all $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ [Saitoh, 1988].

As an aside, note that it is not necessary for the input patterns to come from a vector space. Arbitrary sets of objects can be used, as long as they lead to positive matrices [Schölkopf, 1997]. Indeed, it was shown that one can define kernels

which measure the similarity of highly structured objects with respect to underlying generative models (Jaakkola and Haussler [1999b], cf. Chapters 3 and 4).

From (1.67), it is straightforward to construct a map Φ into a potentially infinite-dimensional l_2 space which satisfies (1.62). For instance, we may use

$$\Phi(\mathbf{x}) = (\sqrt{\lambda_1}\psi_1(\mathbf{x}), \sqrt{\lambda_2}\psi_2(\mathbf{x}), \ldots). \tag{1.68}$$

Rather than thinking of the feature space as an l_2 space, we can alternatively represent it as the Hilbert space $\mathcal{H}_k$ containing all linear combinations of the functions $f(.) = k(\mathbf{x}_i, .)$ $(\mathbf{x}_i \in \mathcal{X})$. To ensure that the map $\Phi : \mathcal{X} \to \mathcal{H}_k$, which in this case is defined as

$$\Phi(\mathbf{x}) = k(\mathbf{x}, .), \tag{1.69}$$

satisfies (1.62), we need to endow $\mathcal{H}_k$ with a suitable dot product $\langle ., . \rangle$. In view of the definition of Φ, this dot product needs to satisfy

$$\langle k(\mathbf{x}, .), k(\mathbf{x}', .) \rangle = k(\mathbf{x}, \mathbf{x}'), \tag{1.70}$$

reproducing kernel

which amounts to saying that k is a *reproducing kernel* for $\mathcal{H}_k$. For a Mercer kernel (1.67), such a dot product does exist. Since k is symmetric, the ψ_i $(i = 1, \ldots, N_{\mathcal{F}})$ can be chosen to be orthogonal with respect to the dot product in $L_2(\mathcal{X})$, i.e., $(\psi_j, \psi_n)_{L_2(\mathcal{X})} = \delta_{jn}$, using the Kronecker δ_{jn}. From this, we can construct $\langle ., . \rangle$ such that

$$\langle \sqrt{\lambda_j}\psi_j, \sqrt{\lambda_n}\psi_n \rangle = \delta_{jn}. \tag{1.71}$$

Substituting (1.67) into (1.70) then proves the desired equality (for further details, see Aronszajn [1950], Wahba [1973], Schölkopf [1997], Girosi [1998]).

sigmoid kernel

Besides (1.63), SV practictioners use sigmoid kernels

$$k(\mathbf{x}, \mathbf{x}') = \tanh(\kappa(\mathbf{x} \cdot \mathbf{x}') + \Theta) \tag{1.72}$$

for suitable values of gain κ and threshold Θ, and radial basis function kernels, as

Gaussian RBF kernel

for instance [Aizerman et al., 1964, Boser et al., 1992, Schölkopf et al., 1997]

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\|\mathbf{x} - \mathbf{x}'\|^2/(2\,\sigma^2)\right), \tag{1.73}$$

with $\sigma > 0$. Note that when using Gaussian kernels, for instance, the feature space $\mathcal{H}_k$ thus contains all superpositions of Gaussians on $\mathcal{X}$ (plus limit points), whereas by definition of Φ (1.69), only single bumps $k(\mathbf{x}, .)$ do have pre-images under Φ.

The main lesson from the study of kernel functions, is that the use of kernels can turn any algorithm that only depends on dot products into a nonlinear algorithm which is linear in feature space. In the time since this was explicitly pointed out [Schölkopf et al., 1998b] a number of such algorithms have been proposed; until then, the applications of the kernel trick were a proof of the convergence of rbf network training by Aizerman et al. [1964] and the nonlinear variant of the SV algorithm by Boser et al. [1992] (see Figure 1.3).

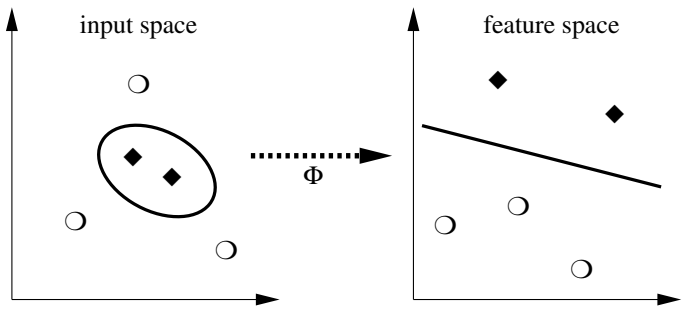


Figure 1.3 The idea of SV machines: map the training data nonlinearly into a higher-dimensional feature space via Φ, and construct a separating hyperplane with maximum margin there. This yields a nonlinear decision boundary in input space. By the use of a kernel function (1.62), it is possible to compute the separating hyperplane without explicitly carrying out the map into the feature space.

To construct SV machines, one computes an optimal hyperplane in feature space. To this end, we substitute $\Phi(\mathbf{x}_i)$ for each training example $\mathbf{x}_i$. The weight vector (cf. (1.56)) then becomes an expansion in feature space. Note that $\mathbf{w}$ will typically no more correspond to the image of just a single vector from input space (cf. Schölkopf et al. [1999b] for a formula to compute the pre-image if it exists), in other words, $\mathbf{w}$ may not be directly accessible any more. However, since all patterns only occur in dot products, one can substitute Mercer kernels k for the dot products [Boser et al., 1992, Guyon et al., 1993], leading to decision functions of the more general form (cf. (1.60))

decision function

$$g(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \left(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_i)\right) + b\right) = \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \, k(\mathbf{x}, \mathbf{x}_i) + b\right) \tag{1.74}$$

and the following quadratic program (cf. (1.58)):

$$\text{maximize} \qquad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \tag{1.75}$$

$$\text{subject to} \qquad \alpha_i \geq 0, \quad i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{1.76}$$

Recall that, as discussed in Section 1.1.4 a separating hyperplane may not always exist, even in the expanded feature space $\mathcal{F}$. To cope with this difficulty, slack variables were introduced to yield the *soft margin* optimal hyperplane problem (1.25). Incorporating kernels, and rewriting (1.25) in terms of Lagrange multipliers, this again leads to the problem of maximizing (1.75), but now subject to the constraints

soft margin and kernels

$$0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m, \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0. \tag{1.77}$$

The only difference from the separable case (1.76) is the upper bound C (to be chosen by the user) on the Lagrange multipliers α_i. This way, the influence of the individual patterns (which could always be outliers) gets limited. As above, the solution takes the form (1.74). The threshold b can be computed by exploiting the fact that for all SVs $\mathbf{x}_i$ with $\alpha_i < C$, the slack variable ξ_i is zero (this again follows from the Karush-Kuhn-Tucker complementarity conditions), and hence

$$\sum_{j=1}^{m} y_j \alpha_j \, k(\mathbf{x}_i, \mathbf{x}_j) + b = y_i. \tag{1.78}$$

The soft margin algorithm can be modified such that it does not require the regularization constant C. Instead, one specifies an upper bound $0 < \nu \leq 1$ on the fraction of points allowed to lie in the margin (asymptotically, the number of SVs) [Schölkopf et al., 1998c]. This leaves us with a homogeneous target function made up by the quadratic part of (1.75), and the constraints

$$0 \leq \alpha_i \leq 1, \quad i = 1, \ldots, m, \quad \sum_{i=1}^{m} \alpha_i y_i = 0, \text{ and } \frac{1}{m} \sum_{i=1}^{m} \alpha_i \geq \nu. \tag{1.79}$$

Finally, we note that several generalizations of the SVM algorithm to different learning tasks exist, such as regression estimation [Vapnik, 1995], density estimation [Vapnik, 1998, Weston et al., 1999], as well as the estimation of a density's support and novelty detection [Schölkopf et al., 1999].

1.3.3 Smoothness and Regularization

For kernel-based function expansions, one can show [Smola and Schölkopf, 1998b] that given a regularization operator P mapping the functions of the learning machine into some dot product space, minimization of the regularized risk

$$R_{\text{reg}}(f) := R_{\text{emp}}(f) + \frac{\lambda}{2} \|Pf\|^2 \tag{1.80}$$

regularized risk

(with a regularization parameter $\lambda \geq 0$) can be written as a constrained optimization problem. For particular choices of the loss function, it further reduces to a SV type quadratic programming problem. The latter thus is not specific to SV machines, but is common to a much wider class of approaches. What gets lost in the general case, however, is the fact that the solution can usually be expressed in terms of a small number of SVs (cf. also Girosi [1998], who establishes a connection between SV machines and basis pursuit denoising [Chen et al., 1999]). This specific feature of SV machines is due to the fact that the type of regularization and the class of functions that the estimate is chosen from are intimately related [Girosi et al., 1993, Smola and Schölkopf, 1998a, Smola et al., 1998b]: the SV algorithm is equivalent to minimizing the regularized risk $R_{\text{reg}}(f)$ on the set of functions

$$f(\mathbf{x}) = \sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b, \tag{1.81}$$

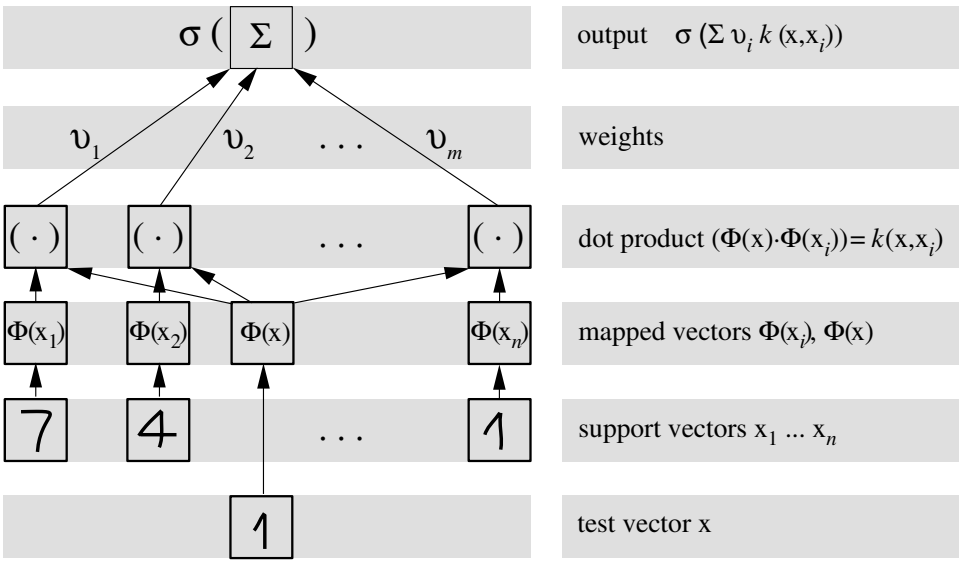


Figure 1.4 Architecture of SV machines. The input $\mathbf{x}$ and the Support Vectors $\mathbf{x}_i$ are nonlinearly mapped (by Φ) into a feature space $\mathcal{F}$, where dot products are computed. By the use of the kernel k, these two layers are in practice computed in one single step. The results are linearly combined by weights υ_i, found by solving a quadratic program (in pattern recognition, $\upsilon_i = y_i\alpha_i$; in regression estimation, $\upsilon_i = \alpha_i^* - \alpha_i$). The linear combination is fed into the function σ (in pattern recognition, $\sigma(x) = \mathrm{sgn}\,(x+b)$; in regression estimation, $\sigma(x) = x+b$).

provided that k and P are interrelated by

$$k(\mathbf{x}_i, \mathbf{x}_j) = ((Pk)(\mathbf{x}_i, .) \cdot (Pk)(\mathbf{x}_j, .))\,. \tag{1.82}$$

To this end, k is chosen as a Green's function of P^*P, for in that case, the right hand side of (1.82) equals

$$(k(\mathbf{x}_i, .) \cdot (P^*Pk)(\mathbf{x}_j, .)) = (k(\mathbf{x}_i, .) \cdot \delta_{\mathbf{x}_j}(.)) = k(\mathbf{x}_i, \mathbf{x}_j). \tag{1.83}$$

For instance, an RBF kernel corresponds to regularization with a functional containing a specific differential operator.

In SV machines, the kernel thus plays a dual role: firstly, it determines the class of functions (1.81) that the solution is taken from; secondly, via (1.82), the kernel determines the type of regularization that is used. Using bounds on covering numbers of Hilbert spaces [Carl and Stephani, 1990], one can show [Williamson et al., 1998, 1999, Schölkopf et al., 1999] that the spectrum of the matrix $(k(x_i, x_j))_{ij}$ is closely connected to the generalization performance and also to the spectrum of the kernel k. This indicates what type of regularization (i.e., kernel) should be used.

regularization networks

For arbitrary expansions of f into basis functions, say f_i, the considerations about smoothness of the estimate still hold, provided $\|Pf\|$ is a norm in the space spanned by the basis functions f_i (otherwise one could find functions $f \in \operatorname{span}\{f_i\}$ with $\|Pf\| = 0$, however $f \neq 0$). In this case the existing bounds for kernel expansions can be readily applied to regularization networks as well (cf., e.g., [Williamson et al., 1998, Smola, 1998] for details). However, one can show [Kimeldorf and Wahba, 1971, Cox and O'Sullivan, 1990], that such an expansion may not fully minimize the regularized risk functional (1.80). This is one of the reasons why often only kernel expansions are considered.

Gaussian processes

Finally it is worth while pointing out the connection between Gaussian Processes and Support Vector machines. The similarity is most obvious in regression, where the Support Vector solution is the maximum a posteriori estimate of the corresponding Bayesian inference scheme [Williams, 1998]. In particular, the kernel k of Support Vector machines plays the role of a covariance function such that the prior probability of a function $f = \sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x})$ is given by

$$P(f) \propto \exp\left(-\tfrac{1}{2}\|Pf\|^2\right) = \exp\left(-\tfrac{1}{2}\sum_{i,j}\alpha_i\alpha_j k(\mathbf{x}_i, \mathbf{x}_j)\right). \tag{1.84}$$

Bayesian methods, however, require averaging over the posterior distribution $P(f|X, Y)$ in order to obtain the final estimate and to derive error bounds. In classification the situation is more complicated, since we have Bernoulli distributed random variables for the labels of the classifier. See [Williams, 1998] for more details on this subject.

1.3.4 A Bound on the Leave-One-Out Estimate

Besides the bounds directly involving large margins, which are useful for stating uniform convergence results, one may also try to estimate $R(f)$ by using leave-one-out estimates. Denote by f_i the estimate obtained from $X\backslash\{\mathbf{x}_i\}, Y\backslash\{y_i\}$. Then

$$R_{\text{out}}(f) := \frac{1}{m}\sum_{i=1}^{m} c(\mathbf{x}_i, y_i, f_i(\mathbf{x}_i)) \tag{1.85}$$

One can show (cf., e.g., [Vapnik, 1979]) that the latter is an unbiased estimator of $R(f)$. Unfortunately, $R_{\text{out}}(f)$ is hard to compute and thus rarely used. In the case of Support Vector classification, however, an upper bound on $R_{\text{out}}(f)$ is not too difficult to obtain. Vapnik [1995] showed that the fraction of Support Vectors is an upper bound on $R_{\text{out}}(f)$. Jaakkola and Haussler [1999b] have generalized this result as follows

$$\begin{aligned} R_{\text{out}}(f) &\leq \frac{1}{m}\sum_{i=1}^{m} 1_{\{y_i \sum_{j\neq i}\alpha_j y_j k(\mathbf{x}_j,\mathbf{x}_i) + y_i b > 0\}} \\ &= \frac{1}{m}\sum_{i=1}^{m} 1_{\{y_i(f(\mathbf{x}_i) - \alpha_i k(\mathbf{x}_i,\mathbf{x}_i)) > 0\}}. \end{aligned} \tag{1.86}$$

The latter can be obtained easily without explicitly solving the optimization problem again for the reduced samples. In particular, for kernels with $k(\mathbf{x}, \mathbf{x}) = 1$ like many RBF kernels the condition reduces to testing whether $y_i f(\mathbf{x}_i) - \alpha_i > 0$. The remaining problem is that $R_{\text{out}}(f)$ itself is a random variable and thus it does not immediately give a *bound* on $R(f)$. See also Chapters 15 and 17 for futher details on how to exploit these bounds in practical cases.

1.4 Boosting

Freund and Schapire [1997] proposed the AdaBoost algorithm for combining classifiers produced by other learning algorithms. AdaBoost has been very successful in practical applications (see Section 1.5). It turns out that it is also a large margin technique.

Table 1.2 gives the pseudocode for the algorithm. It returns a convex combination of classifiers from a class G, by using a learning algorithm L that takes as input a training sample X, Y and a distribution D on X (not to be confused with the true distribution p), and returns a classifier from G. The algorithm L aims to minimize training error on X, Y, weighted according to D. That is, it aims to minimize

$$\sum_{i=1}^{m} D_i 1_{\{h(\mathbf{x}_i) \neq y_i\}}. \tag{1.87}$$

AdaBoost iteratively combines the classifiers returned by L. The idea behind AdaBoost is to start with a uniform weighting over the training sample, and progressively adjust the weights to emphasize the examples that have been frequently misclassified by the classifiers returned by L. These classifiers are combined with convex coefficients that depend on their respective weighted errors. The following theorem shows that Adaboost produces a large margin classifier, provided L is successful at finding classifiers with small weighted training error. See [Schapire et al., 1998]. Recall (1.30) that the margin error of a function f with respect to ρ on a sample X, Y is $R_\rho(f) = \frac{1}{m} \sum_{i=1}^{m} 1_{\{y_i f(\mathbf{x}_i) < \rho\}}$.

Theorem 1.17 Margin Error of AdaBoost
If, at iteration t, L returns a function with weighted training error $\varepsilon_t < 1/2$, then `AdaBoost` returns a function f that satisfies

$$R_\rho(f) \leq 2^T \prod_{t=1}^{T} \sqrt{\varepsilon_t^{1-\rho}(1-\varepsilon_t)^{1+\rho}}. \tag{1.88}$$

In particular, if $\varepsilon_t \leq 1/2 - 2\rho$, then

$$R_\rho(f) < (1 - \rho^2)^{T/2}, \tag{1.89}$$

and this is less than ε for $T \geq (2/\rho^2) \ln(1/\varepsilon)$.

Algorithm 1.2 : Adaboost

argument: Training sample, $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathcal{X}$, $Y = \{y_1, \ldots, y_m\} \subset \{\pm 1\}$
Number of iterations, T
Learning algorithm L that chooses a classifier from G to minimize the weighted training error.
returns: Convex combination of functions from G, $f = \sum_{t=1}^{T} \alpha_t g_t$.

function AdaBoost(X, Y, T)
  for all i from $i = 1, \ldots, m$
    $D_1(i) := 1/m$
  endfor
  for all t from $\{1, \ldots, T\}$
    $g_t := L(X, Y, D_t)$
    $$\varepsilon_t := \sum_{i=1}^{m} D_t(i) 1_{g_t(x_i) \neq y_i}$$
    $$\alpha_t := \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$
    $$Z_t := 2\sqrt{\varepsilon_t (1 - \varepsilon_t)}$$
    for all i from $i = 1, \ldots, m$
    $$D_{t+1}(i) := \begin{cases} D_t(i) e^{-\alpha_t} / Z_t & \text{if } y_i = g_t(x_i) \\ D_t(i) e^{\alpha_t} / Z_t & \text{otherwise,} \end{cases}$$
    endfor
  endfor
  return $f = \dfrac{\sum_{t=1}^{T} \alpha_t g_t}{\sum_{i=1}^{T} \alpha_t}$.
end

1.5 Empirical Results, Implementations, and Further Developments

Large margin classifiers are not only promising from the theoretical point of view. They also have proven to be competitive or superior to other learning algorithms in practical applications. In the following we will give references to such situations.

1.5.1 Boosting

Experimental results show that boosting is able to improve the performance of classifiers significantly. Extensive studies on the UC Irvine dataset, carried out by Freund and Schapire [1996] and Quinlan [1996a] with tree classifiers show the performance of such methods. However, also other learning algorithms can benefit from boosting. Schwenk and Bengio [1998] achieve record performance on an OCR task on the UC Irvine database, using neural networks as the base classifiers. See Rätsch [1998] and Chapter 12 for further results on the performance of improved versions of boosted classifiers.

1.5.2 Support Vector Machines

SV Machines perform particularly well in feature rich highdimensional problems. Schölkopf et al. [1995], Schölkopf et al. [1997, 1998a] achieve state of the art, or even record performance in several Optical Character Recognition (OCR) tasks such as the digit databases of the United Postal Service (USPS) and the National Institute of Standards and Technology (NIST). The latter can be obtained at

`http://www.research.att.com/~yann/ocr/mnist/`

Similar results have been obtained for face recognition by Oren et al. [1997], Osuna et al. [1997b] and object recognition [Blanz et al., 1996, Schölkopf, 1997]. Finally, also on large noisy problems SV Machines are very competitive as shown in [Smola, 1998, Vannerem et al., 1999].

1.5.3 Implementation and Available Code

Whilst Boosting can be easily implemented by combining a base learner and following the pseudocode of Table 1.2. Hence one only has to provide a base learning algorithm satisfying the properties of a weak learner, which defers all problems to the underlying algorithm.

`http://www.research.att.com/~yoav/adaboost/`

provides a Java applet demonstrating the basic properties of AdaBoost.

The central problem in Support Vector Machines is a quadratic programming problem. Unfortunately, off-the-shelf packages developed in the context of mathematical programming like MINOS [Murtagh and Saunders, 1998], LOQO [Vanderbei, 1994], OSL [IBM Corporation, 1992], or CPLEX [CPL, 1994] are often prohibitively expensive or unsuitable for optimization problems in more than several thousand variables (whilst the number of variables may be in the tens of thousands in practical applications). Furthermore these programs are often optimized to deal with sparse matrix entries, causing unneeded overhead when solving generic SV optimization problems (which are sparse in the solution, not in the matrix entries).

This situation led to the development of several quadratic optimization algorithms specifically designed to suit the needs of SV machines. Starting from simple subset selection algorithms as initially described by Vapnik [1979] and subsequently implemented in, e.g., [Schölkopf et al., 1995], more advanced chunking methods were proposed [Osuna et al., 1997a] (see also [Joachims, 1999] for a detailed description of the algorithm) for splitting up the optimization problem into smaller subproblems that could be easily solved by standard optimization code. Other methods exploit constrained gradient descent techniques [Kaufmann, 1999], or minimize very small subproblems, such as the Sequential Minimal Optimization algorithm (SMO) by Platt [1999]. See also Chapter 6 for further methods for training a SV classifier. Implementations include SvmLight by Joachims [1999],

http://www-ai.cs.uni-dortmund.de/thorsten/svm_light.html,

the Royal Holloway / AT&T / GMD Support Vector Machine by Saunders et al. [1998], available at

http://svm.dcs.rhbnc.ac.uk/,

and the implementation by Steve Gunn which can be downloaded from

http://www.isis.ecs.soton.ac.uk/resources/svminfo/.

The first two of these optimizers use the GMD (Smola) implementation of an interior point code along the lines of Vanderbei [1994] as the core optimization engine. It is available as a standalone package at

http://www.kernel-machines.org/software.html.

This site will also contain pointers to further toolboxes as they become available. Java applets for demonstration purposes can be found at

http://http://svm.dcs.rhbnc.ac.uk/pagesnew/GPat.shtml
http://http://svm.research.bell-labs.com/SVT/SVMsvt.html.

1.6 Notation

We conclude the introduction with a list of symbols which are used throughout the book, unless stated otherwise.

$\mathbb{N}$	the set of natural numbers
$\mathbb{R}$	the set of reals
X	a sample of input patterns
Y	a sample of output labels
$\mathcal{X}$	an abstract domain
ln	logarithm to base e
$\log_2$	logarithm to base 2
$(\mathbf{x} \cdot \mathbf{x}')$	inner product between vectors $\mathbf{x}$ and $\mathbf{x}'$
$\|.\|$	2-norm (Euclidean distance), $\|\mathbf{x}\| := \sqrt{(\mathbf{x} \cdot \mathbf{x})}$
$\|.\|_p$	p-norm , $\|\mathbf{x}\|_p := \left(\sum_{i=1}^N \|x_i\|^p\right)^{1/p}$
$\|.\|_\infty$	∞-norm , $\|\mathbf{x}\|_\infty := \max_{i=1}^N \|x_i\|$
ℓ_p	ℓ_p metric
$L_2(X)$	space of functions on X square integrable wrt. Lebesgue measure
$\mathbf{E}(\xi)$	expectation of random variable ξ
$\Pr(\cdot)$	probability of an event
N	dimensionality of input space
m	number of training examples
$\mathbf{x}_i$	input patterns
y_i	target values, or (in pattern recognition) classes
$\mathbf{w}$	weight vector
b	constant offset (or threshold)
h	VC dimension
f	a real valued function $f : \mathbb{R}^N \to \mathbb{R}$ (unthresholded)
F	a family of real valued functions f
g	a decision function $g : \mathbb{R}^N \to \{-1, 1\}$
F	a family of decision functions g
$\rho_f(\mathbf{x}, y)$	margin of function f on the example $(\mathbf{x}, y)$, i.e., $y\, f(\mathbf{x})$
ρ_f	minimum margin, i.e., $\min_{1 \le i \le m} \rho_f(\mathbf{x}_i, y_i)$

$c(\mathbf{x}, y, f(\mathbf{x}))$	cost function
$R(g)$	risk of g, i.e., expected fraction of errors
$R_{\mathrm{emp}}(g)$	empirical risk of g, i.e., fraction of training errors
$R(f)$	risk of f
$R_{\mathrm{emp}}(f)$	empirical risk of f
k	Mercer kernel
$\mathcal{F}$	Feature space induced by a kernel
Φ	map into feature space (induced by k)
α_i	Lagrange multiplier
$\boldsymbol{\alpha}$	vector of all Lagrange multipliers
ξ_i	slack variables
$\boldsymbol{\xi}$	vector of all slack variables
C	regularization constant for SV Machines
λ	regularization constant ($C = \frac{1}{\lambda}$)

2 Roadmap

Support Vector Machines

Chapter 3 One of the most important issues in current research on SV machines is how to design suitable kernels for specific applications. Problems involving categorical or binary valued data so far constituted a difficult setting for kernel methods. In his chapter, Watkins presents a new concept using generative models to construct **Dynamic Alignment Kernels**. These are based on the observation that the sum of products of conditional probabilities $\sum_c p(\mathbf{x}|c)p(\mathbf{x}'|c)$ is a valid SV kernel. This is particularly well suited for the use of Hidden Markov Models, thereby opening the door to a large class of applications like DNA analysis or speech recognition.

Chapter 4 The contribution of Oliver, Schölkopf, and Smola, deals with a related approach. It analyses **Natural Regularization from Generative Models**, corresponding to a class of kernels including those recently proposed by Jaakkola and Haussler [1999b]. The analysis hinges on information-geometric properties of the log probability density function (generative model) and known connections between support vector machines and regularization theory, and proves that the maximal margin term induced by the considered kernel corresponds to a penalizer computing the L_2 norm weighted by the generative model. Moreover, it is shown that the feature map corresponding to the kernel whitens the data.

Chapter 5 Large margin classifiers such as SV machines may be good for correct classification, however lack a practical means to give a probabilistic interpretation of a classifier's output, i.e., a confidence rating. This problem is addressed by Platt by fitting a logistic to the function values of a SVM in order to obtain **Probabilities for SV Machines**. The results are comparable to classical statistical techniques such as logistic regression while conserving the sparseness and thus numerical efficiency of SVMs. Pseudocode is given for easy implementation.

Chapter 6 Kowalczyk presents an in-depth overview of sequential update algorithms for the **Maximal Margin Perceptron**. In particular, he derives a new update method which is based on the observation that the normal vector of the separating hyperplane can be found as the difference between between two points lying in the convex hull of positive and negative examples respectively. This new method has the advantage that at each iteration only one Lagrange multiplier has to be updated, leading to a potentially faster training algorithm. Bounds on the speed of convergence are stated and an experimental comparison with other training algorithms shows the good performance of this method.

Chapter 7 Based on ideas from SV classification, Herbrich, Graepel, and Obermayer construct an algorithm to obtain **Large Margin Rank Boundaries for Ordinal Regression**. In other words, they present a SV algorithm for learning preference relations. In addition to that, the chapter contains a detailed derivation of the corresponding cost functions, risk functionals, and proves uniform convergence bounds for the setting. This could be useful for other classes of large margin learning algorithms, too. Experimental evidence shows the good performance of their distribution independent approach.

Kernel Machines

Chapter 8 Arbitrary kernel functions which need not satisfy Mercer's condition can be used by the **Generalized Support Vector Machines** algorithm, presented by Mangasarian. This goal is achieved by separating the regularizer from the actual separation condition. For quadratic regularization this leads to a convex quadratic program that is no more difficult to solve than the standard SV optimization problem. Sparse expansions are achieved when the 1-norm of the expansion coefficients is chosen to restrict the class of admissible functions. The problems are formulated in a way which is compatible with Mathematical Programming literature.

Chapter 9 In their chapter on **Linear Discriminant and Support Vector Classifiers**, Guyon and Stork give a thorough and authoritative review of linear discriminant algorithms. SVMs in feature space are one special case of this, and Guyon and Stork point out similarities and differences to other cases. Placing SVMs into this wider context provides a most useful backdrop which should help avoiding SVM specialist discussions losing sight of the general picture.

Chapter 10 The connection between **Regularization Networks and Support Vector Machines** is explored by Evgeniou, Pontil, and Poggio. They review uniform convergence results for such learning techniques, and present a new theoretical justification of SVM and Regularization Networks based on Structural Risk Minimization. Furthermore, they give an overview over the current state of the art regarding connections between Reproducing Kernel Hilbert Spaces, Bayesian Priors, Feature Spaces and sparse approximation techniques.

Boosting

Chapter 11 In their chapter on **Robust Ensemble Learning**, Rätsch, Schölkopf, Smola, Mika, Onoda, and Müller propose two new voting methods that are more robust to noise than AdaBoost and related algorithms. These algorithms are inspired by the observation that voting methods such as AdaBoost can be viewed as finding approximate solutions to a linear program. Rather than relying on the weight of a regularization term, the algorithms use a parameter ν that is akin to an estimate of the noise level in the data (the proportion of training errors). These algorithms have the attractive property that they produce an estimate of the effective complexity of the combined classifier.

Chapter 12 Mason, Baxter, and Bartlett then present an elegant generalization of boosting algorithms in their chapter on **Functional Gradient Techniques for Combining Hypotheses**. Here they view classifier voting procedures as, abstractly, performing iterative descent over an inner product space, and show how existing voting methods such as AdaBoost can be obtained as special cases of these more general procedures. Mason *et al.* then show how the training convergence of existing methods follows as a special case of a much more general convergence analysis. The main practical contribution of this chapter is the introduction of a new (sigmoidal) margin cost functional that can be optimized by a heuristic search procedure (DOOM II). The resulting procedure achieves good theoretical bounds on its generalization performance but also demonstrates systematic improvements over AdaBoost in empirical tests—especially in domains with significant classification noise.

Chapter 13 In their chapter entitled **Towards a Strategy for Boosting Regressors**, Karakoulas and Shawe-Taylor describe a new strategy for combining regressors (as opposed to classifiers) in a boosting framework. They base their approach on a soft margin generalization error bound which is expressed in terms of a given loss measure. Karakoulas and Shawe-Taylor derive a boosting procedure that iteratively minimizes this loss, and obtain a novel strategy for weighting the training examples and determining their target values (suitable for regression problems). Their resulting procedure demonstrates promising improvements in generalization performance over earlier *ad hoc* approaches in empirical tests.

Leave-One-Out Methods

Chapter 14 Vapnik and Chapelle present **Bounds on the Error Expectation for SVM** in terms of the leave-one-out estimate and the expected value of certain properties of the SVM. In their work, which follows up on an announcement by Vapnik during the workshop that the present volume is based on, they show that previous bounds involving the minimum margin and the diameter D of the set of support vectors can be improved by the replacement of D^2 by SD. Here, S is a new geometric property of the support vectors that Vapnik and Chapelle call the span. Experimental results show that this improvement gives significantly better predictions of test error than the previous bounds, and seems likely to be useful for model selection.

Chapter 15 In their contribution **Adaptive Margin Support Vector Machines**, Weston and Herbrich take the converse approach. Based on a leave-one-out bound of Jaakkola and Haussler [1999b], they devise a modification of the original SV algorithm in order to minimize the bound directly. This formulation is essentially parameter free, maintains sparsity of the solution, and can be solved by a linear program. The novelty can be found in the fact that rather than maximizing the overall minimum margin, the individual margin of patterns is maximized adaptively. Experiments show that its classification performance is very competitive with an optimally adjusted SV machine and comparable to a ν-SV classifier. Uniform convergence bounds are provided.

Chapter 16 Wahba, Lin and Zhang introduce **Generalized Approximate Cross Validation (GACV) for Support Vector Machines**. They view SVMs as a regularization technique in a reproducing kernel Hilbert space. They review the generalized comparative Kullback-Leibler distance (GCKL) and they show that the GCKL for the SVM is an upper bound on its expected misclassification rate. They derive the GACV as an estimate of the GCKL, as a function of certain tunable parameters. Preliminary simulations suggest that the GACV has application to model selection problems, since the minimizer of the GACV is a reasonable estimate of the minimizer of the GCKL.

Chapter 17 The study on **Gaussian Processes and SVM: Mean Field and Leave-One-Out** gives an overview of the connections between Gaussian Processes and SV machines and the implications for cost functions and corresponding probabilistic settings. The authors, Opper and Winther, use the 'naive mean field' approximation from Statistical Mechanics to provide estimates on the leave-one-out error in kernel methods which are fast to compute and in very good agreement with the true leave-one-out error. Experimental results corroborate this finding.

Beyond the Margin

Chapter 18 Ruján and Marchand propose an algorithm for **Computing the Bayes Kernel Classifier**. In the so-called version space view of classification, the SVM solution of a separable learning problem corresponds to the center of the largest inscribable sphere in a polytope determined by the training examples. Statistically, however, it would be preferable to find a solution corresponding to the center of mass. Ruján and Marchand propose a Billiard algorithm which, under the assumption of ergodicity, converges towards the latter.

Chapter 19 Rather than considering the minimum margin, Shawe-Taylor and Cristianini focus on **Margin Distribution and Soft Margin**. The latter is a more robust quantity than the minimum margin itself which can be easily decreased by a single mislabeled example. In particular they provide provide generalization bounds, which motivate algorithms maximizing the minimum margin plus the 2-norm of the slack variables for those patterns violating the margin condition. This is not the standard setting in SV machines which in general use the 1-norm of the slacks, however, it coincides with the target function of optimization algorithms such as the one in Chapter 6 and can be useful in this regard.

Chapter 20 In their chapter on **Support Vectors and Statistical Mechanics**, Dietrich, Opper, and Sompolinsky analyze SVMs using methods of statistical mechanics by representing the SVM solution as the limit of a family of Gibbs distributions. This way, they are able to derive rather precise learning curves. Their analysis shows that for "favourable" input distributions, i.e., ones which allow a large margin, the expected generalization error decays much more rapidly than predicted by distribution-independent upper bounds of statistical learning theory.

Chapter 21

Section 1.2.3 explained the role of covering numbers of classes of real-valued functions in generalization error bounds. Smola, Elisseeff, Schölkopf, and Williamson present bounds on the **Entropy Numbers for Convex Combinations and MLPs**. These bounds improve on previous results for convex combinations of parameterized functions (such as combinations of classifiers) and compositions of these combinations (such as multi-layer neural networks). In the latter case especially, the new bounds presented in Chapter 21 are substantially smaller than the previous results. They show that even more substantial improvements are possible when the parameterized functions involve kernels with rapidly decreasing eigenvalues. This gives the best known bounds for the covering numbers of radial basis function networks, for instance.

I Support Vector Machines

3 Dynamic Alignment Kernels

Chris Watkins
Royal Holloway College
University of London
Department of Computer Science,
Egham, Surrey, TW20 OEX, UK
chrisw@dcs.rbnc.ac.uk

There is much current interest in kernel methods for classification, regression, PCA, and other linear methods of data analysis. Kernel methods may be particularly valuable for problems in which the input data is not readily described by explicit feature vectors. One such problem is where input data consists of symbol-sequences of different lengths and the relationships between sequences are best captured by dynamic alignment scores.

This paper shows that the scores produced by certain dynamic alignment algorithms for sequences are in fact valid kernel functions. This is proved by expressing the alignment scores explicitly as scalar products.

Dynamic alignment kernels are potentially applicable to biological sequence data, speech data, and time series data.

3.1 Introduction: Linear Methods using Kernel Functions

In many types of machine learning, the learner is given a training set of *cases* or *examples* $\mathbf{x}_1 \ldots \mathbf{x}_l \in \mathcal{X}$, where $\mathcal{X}$ denotes the set of all possible cases: cases may be vectors, pieces of text, biological sequences, sentences, and so on. For supervised learning, the cases are accompanied by corresponding *labels* or *values* $y_1 \ldots y_l$. The cases are mapped to *feature vectors* $\mathbf{v}_1 \ldots \mathbf{v}_l \in \mathcal{F}$, where $\mathcal{F}$ is a real finite or Hilbert space termed the *feature space*. The mapping from $\mathcal{X}$ to $\mathcal{F}$ is denoted by ϕ, so that $\mathbf{v}_i = \phi(\mathbf{x}_i)$. Sometimes the cases are given as feature vectors to start with, in which case ϕ may be the identity mapping; otherwise ϕ denotes the method of assigning numeric feature values to a case.

Once a feature vector $\mathbf{v}_i$ has been defined for each case $\mathbf{x}_i$, it becomes possible to apply a wide range of linear methods such as support-vector machines, linear regression, principal components analysis (PCA), and k-means cluster analysis.

As shown in [Boser et al., 1992] for SV machines, in for example [Wahba, 1990] for linear regression, and in [Schölkopf et al., 1998b] for PCA and k-means cluster analysis, the calculations for all of these linear methods may be carried out using a dual rather than a primal formulation of the problem.

For example, in linear least-squares regression the primal formulation is to find a coefficient vector $\boldsymbol{\beta}$ that minimises $\|X\boldsymbol{\beta} - \mathbf{y}\|$ where X is the design matrix. If there are d features, this is an l by d matrix in which the ith row is $\mathbf{v}_i$, and each $\mathbf{v}_i$ has d elements. If l is larger than d, the usual method of finding $\boldsymbol{\beta}$ is to solve the normal equations $X^T X\boldsymbol{\beta} = X^T\mathbf{y}$. This requires the solution of a set of linear equations with coefficients given by the $d \times d$ matrix $X^T X$.

The dual formulation is to find a vector $\boldsymbol{\alpha}$ that minimises $\|XX^T\boldsymbol{\alpha} - y\|$, so that one coefficient α_i is found for each case vector $\mathbf{x}_i$. This requires the solution of a set of linear equations with coefficients given by the $l \times l$ matrix XX^T.

Both methods lead to the same predicted value $\hat{y}$ for a new case $\mathbf{x}$. If there are more cases than features, that is if $l > d$, the primal method is more economical because the $d \times d$ matrix $X^T X$ is smaller than the $l \times l$ matrix XX^T. For example, if there are 200 cases, each described by a vector of 10 measurements, then the primal method requires solving a 10 by 10 system of linear equations, while the dual method requires solving a 200 by 200 system, which will have rank at most 10. For such a problem, the dual method has no advantage.

The potential advantage of the dual method for regression is that it can be applied to very large feature vectors. The coefficient matrix XX^T contains the scalar products of pairs of feature vectors: the ijth element of XX^T is $\mathbf{v}_i \cdot \mathbf{v}_j$. In the dual calculation, it is only scalar products of feature vectors that are used—feature vectors never appear on their own. The matrix of scalar products of the feature vectors encodes the lengths and relative orientations of the features, and this geometric information is enough for most linear computations.

computational advantage of kernel method

As the feature vectors $\mathbf{v}_i = \phi(\mathbf{x}_i)$ appear only in scalar products, it is often possible to avoid computing the feature vectors, and to compute scalar products directly in some economical fashion from the case descriptions $\mathbf{x}_i$ instead. A *kernel* is a function k that accepts two case descriptions as arguments, and computes the scalar product of the corresponding feature vectors.

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x}) \cdot \phi(\mathbf{x}') \tag{3.1}$$

The feature-space mapping ϕ determines k uniquely, but k determines only the metric properties of the image under ϕ of the case-set $\mathcal{X}$ in feature space. ϕ is not in general invertible, and indeed $\phi(\mathcal{X})$ need not even be a linear subspace of $\mathcal{F}$. ϕ need not be and in general is not a linear mapping: indeed, addition and multiplication need not even be defined for elements of $\mathcal{X}$, if, for example, they are strings.

The dual formulation often has a computational advantage over the primal formulation if the kernel function k is easy to compute, but the mapping to feature space ϕ is infeasible to compute. A well-known example of this is the "homogeneous polynomial kernel" of [Vapnik, 1995] in which the cases $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ are real d dimensional vectors:

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^n \tag{3.2}$$

$$= \sum_{i_1=1}^{d} \cdots \sum_{i_n=1}^{d} (x_{i_1} \cdots x_{i_n}) (z_{i_1} \cdots z_{i_n}) \tag{3.3}$$

for some positive integer n, and $1 \leq i_1, \ldots, i_n \leq d$. A mapping ϕ that induces this kernel is, for $\mathbf{x} = \langle x_1, \ldots, x_d \rangle$

$$\phi(\mathbf{x}) = \langle x_{i_1} \cdots x_{i_n} : 1 \leq i_1 \ldots i_n \leq d \rangle \tag{3.4}$$

In the character recognition application described in [Vapnik, 1995], the cases were vectors with dimension 256 and values of n up to 8 were used, so that the vectors in (3.4) had billions of terms, and the expression (3.2) was vastly easier to compute than the explicit scalar product (3.3).

3.2 Applying Linear Methods to Structured Objects

Not all data comes naturally as vectors: data may consist of "structured objects," such as sequences of different lengths, trees, or sentences. To apply linear methods to such data, it is necessary either to construct feature vectors explicitly, or to use a kernel function. The recent success of the methods of [Joachims, 1998] in text classification has shown how valuable it can be to apply linear statistical methods to inductive problems where such methods have not previously been used. This section describes three approaches to mapping structured objects to vectors in order to apply linear statistical methods.

3.2.1 Sparse Vector Kernels

Joachims [1998] considered the problem of classifying text news stories by subject. Essentially, Joachims considered a text as a sparse vector, with one dimension for each possible word. With an efficient sparse representation, the dot-product of two sparse vectors can be computed in a time proportional to the total number of non-zero elements in the two vectors. A kernel implemented as a sparse dot-product is a natural method of applying linear methods to sequences. Examples of such sparse-vector mappings are:

text classification

- mapping a text to the set of words it contains
- mapping a text to the set of pairs of words that are in the same sentence
- mapping a symbol sequence to the set of all subsections of some fixed length m

"Sparse-vector kernels" are an important extension of the range of applicability of linear methods.

3.2.2 Case-based Features

Often, there are natural matching functions or similarity scores that may be applied to structured objects. These are functions that can be applied to a pair of objects, and which return a real-valued score. Although such a matching is not necessarily representable as a scalar product, any such function can be used to create features in the following way.

Given *any* function $f : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, and an indexed set of cases, $\mathbf{x}_1, \ldots, \mathbf{x}_n$ a possible feature space mapping is

$$\phi(\mathbf{x}) = \langle f(\mathbf{x}_1, \mathbf{x}), \ldots, f(\mathbf{x}_n, \mathbf{x}) \rangle \tag{3.5}$$

This is not really a true kernel method, as the feature vector is computed explicitly, and there is no computational advantage in using a kernel. For further details on this type of map, and the construction of the kernel corresponding to f, cf. [Schölkopf et al., 1999b].

3.2.3 Diagonal-dominance Kernels

A second canonical construction for a kernel k given any $f : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$, for a finite or countable set $\mathcal{X}$, uses a feature space with dimensions indexed by $\mathcal{X} \times \mathcal{X}$, and for any $\mathbf{x} \in \mathcal{X}$ the $\langle \mathbf{a}, \mathbf{b} \rangle$th element of the vector $\phi(\mathbf{x})$ is defined as

$$[\phi(\mathbf{x})]_{\langle \mathbf{a}, \mathbf{b} \rangle} = \begin{cases} f(\mathbf{a}, \mathbf{b}) & \text{if } \mathbf{a} = \mathbf{x} \text{ or } \mathbf{b} = \mathbf{x} \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

so that k is defined as

$$k(\mathbf{a}, \mathbf{b}) = f(\mathbf{a}, \mathbf{b})^2 + f(\mathbf{b}, \mathbf{a})^2 \quad \text{if } \mathbf{a} \neq \mathbf{b} \tag{3.7}$$

and

$$k(\mathbf{a}, \mathbf{a}) = f(\mathbf{a}, \mathbf{a})^2 + \sum_{\mathbf{c} \in \mathcal{X}, \mathbf{c} \neq \mathbf{a}} f(\mathbf{a}, \mathbf{c})^2 + \sum_{\mathbf{c} \in \mathcal{X}, \mathbf{c} \neq \mathbf{a}} f(\mathbf{c}, \mathbf{a})^2 \tag{3.8}$$

This "diagonal-dominance" kernel does in some sense provide a computational advantage, for it enables an arbitrary non-negative symmetric function $k(\mathbf{x}, \mathbf{z}) = f(\mathbf{x}, \mathbf{z})^2 + f(\mathbf{z}, \mathbf{x})^2$ for $\mathbf{x} \neq \mathbf{z}$ to be used as a kernel, provided that the diagonal entries $k(\mathbf{x}, \mathbf{x})$ are made sufficiently large that any finite matrix of dot-products of distinct elements of $\mathcal{X}$ will be diagonally dominant, and therefore positive semidefinite.

The size of diagonal element required may be reduced by defining ϕ with respect to a reference data set $\mathcal{R} \subset \mathcal{X}$

$$[\phi(\mathbf{x})]_{\langle \mathbf{a}, \mathbf{b} \rangle} = \begin{cases} f(\mathbf{a}, \mathbf{b}) & \text{if } (\mathbf{a} = \mathbf{x} \text{ or } \mathbf{b} = \mathbf{x}) \text{ and } (\mathbf{a} \in \mathcal{R} \text{ or } \mathbf{b} \in \mathcal{R}\ ) \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

If $\mathcal{R}$ is taken to be a small subset of $\mathcal{X}$—perhaps the training data set itself—then the diagonal elements of the matrix of dot-products of the training data can be set to the sums of the rows. The diagonal elements from (3.9) may be much smaller than those from (3.6). It is curious that this construction of an explicit dot-product for a diagonally dominant matrix only works for matrices with non-negative elements.

Unfortunately matrices with large diagonal elements are likely to provide poor generalization in learning. Nevertheless, this construction may sometimes be of use.

3.3 Conditional Symmetric Independence Kernels

Joint probability distributions are often used as scoring functions for matching: two objects "match" if they are in some sense similar, and the degree of similarity or relatedness is defined according to a joint probability distribution that assigns pairs of related objects higher probabilities than pairs of unrelated objects. A joint p.d. used in this way will be described in section Section 3.4 below. It is sometimes possible to show that such a joint p.d. is a valid kernel by showing that the p.d. is conditionally symmetrically independent.

Definition 3.1
A joint probability distribution is *conditionally symmetrically independent* (CSI) if it is a mixture of a finite or countable number of symmetric independent distributions.

CSI joint probability distributions may be written as scalar products in the following way. Let X, Z be two discrete random variables, and let p be the joint distribution function, defined as

$$p(x, z) = Pr(X = x \text{ and } Z = z) \tag{3.10}$$

and let p be symmetric—that is, $p(x, z) = p(z, x)$ for all x, z. Let C be a random variable such that

$$Pr(X, Z \mid C) = Pr(X \mid C)Pr(Z \mid C) \tag{3.11}$$

and, given C, the distributions of X and Z are identical. Then

$$p(x, z \mid c) = p(x \mid c)p(z \mid c) \tag{3.12}$$

CSI kernel definition

for each c in the range $\mathcal{C}$ of C ($\mathcal{C}$ is the set of values that C may take). Then

$$\begin{aligned} p(x, z) &= \sum_c p(x \mid c)p(z \mid c)p(c) \\ &= \sum_c \left(p(x \mid c)\sqrt{p(c)}\right)\left(p(z \mid c)\sqrt{p(c)}\right) \end{aligned} \tag{3.13}$$

CSI feature space mapping

where c takes all values in the range of C. This is a scalar product, with the feature-

space mapping defined as

$$\phi(x) = \left\langle p(x \mid c)\sqrt{p(c)} \,:\, c \in \mathcal{C} \right\rangle \tag{3.14}$$

so that

$$p(x, z) = \phi(x) \cdot \phi(z) \tag{3.15}$$

We believe that this definition can be extended to benign cases in which p is a probability density which is a mixture of an uncountable number of symmetric independent densities, indexed by some real-valued parameter c. The technical complications of such an extension are beyond the scope of this paper.

It is evident that any CSI joint p.d. must be positive semidefinite, but we are so far unable to establish whether the converse holds, even in the finite-dimensional case. That is, we do not know whether all positive semidefinite finite joint probability distributions are CSI.

3.4 Pair Hidden Markov Models

A pair hidden Markov model (PHMM) is an HMM that generates two symbol sequences simultaneously; the two sequences need not necessarily be of the same length. The PHMM, therefore, defines a joint probability distribution over finite symbol sequences. Models of this type are used in bioinformatics to construct probabilistic models of relatedness of pairs of protein or DNA sequences. Durbin et al. [1998] provide an excellent tutorial introduction and review of HMMs, PHMMs, and their use in biological sequence analysis.

A PHMM is defined as follows.

- a finite set S of states, which is the disjoint union of four subsets:

S^{AB} — states that emit two symbols, one for A and one for B

$\mathcal{S}^{A}$ — states that emit one symbol only for A

$\mathcal{S}^{B}$ — states that emit one symbol only for B

$\mathcal{S}^{-}$ — states that emit no symbols

- Distinguished states START and END. The process starts in START, and ends in the absorbing state END. For notational reasons, it will be convenient to define that START, END $\in \mathcal{S}^{AB}$, but both START and END emit no symbols.
- A function $T^{\star}$ that gives state transition probabilities: $T^{\star}(\mathsf{s}, \mathsf{t})$ is the probability that the next state is t given that the current state is s.
- An alphabet $\mathcal{B}$
- For states that emit symbols, probability distributions over $\mathcal{B}$:
  - For each state $\mathsf{s} \in \mathcal{S}^{AB}$, a probability distribution over $\mathcal{B} \times \mathcal{B}$
  - For each state $\mathsf{s} \in \mathcal{S}^{A}$or $\mathcal{S}^{B}$ a probability distribution over $\mathcal{B}$

The class $\mathcal{S}^-$ of non-emitting states is included for notational convenience: all states in $\mathcal{S}^-$ can be eliminated with no change to the joint distribution of emitted sequences.

A *realization* of the PHMM is a sequence of states, starting with START and finishing with END, together with the symbol(s), if any, emitted in each state. Each realization, therefore, is a complete record of the construction of two particular sequences $\mathbf{a}$ and $\mathbf{b}$.

The probability of any one realization of a PHMM is straightforward to calculate — see Section 3.4.1 below. But any particular pair of sequences $\mathbf{a}$ and $\mathbf{b}$ may be generated by exponentially many different realizations. Happily there are well-known efficient dynamic programming algorithms for summing over all possible realizations to calculate the joint probability of any two sequences $\mathbf{a}$ and $\mathbf{b}$.

The point of using a PHMM is that it is easy to compute joint probabilities of pairs of sequences. Under what circumstances can this joint probability be represented as a scalar product and used as a kernel?

3.4.1 Computing the Joint Probability of Two Sequences

Pair hidden Markov models are of practical importance because there is an efficient dynamic programming algorithm to calculate the joint probability of a pair of sequences, as follows.

For a state s that emits just one symbol, let the probability that it emits one symbol a to the first sequence only be written $E(\mathsf{s}, a, \cdot)$; let $E(\mathsf{s}, \cdot, b)$ be defined similarly. If, for example, $\mathsf{s} \in \mathcal{S}^A$, then $E(\mathsf{s}, \cdot, b)$ and $E(\mathsf{s}, a, b)$ are both zero; ony $E(\mathsf{s}, a, \cdot)$ may be positive. Each state will emit a symbol eith always to the first string, or always to the second string, or always two symbols, one to each string.

If s emits two symbols, let the probability of emitting the pair a, b to the first and second sequence respectively be $E(\mathsf{s}, a, b)$. Symbol emissions and state transitions are independent: given that the current state is either the start state or an emitting state s, let the probability that the next emitting state (or the end state) is t be written $T(\mathsf{s}, \mathsf{t})$ (in this way we ignore non-emitting states). The joint probability of emitting two sequences $\mathbf{a}$ and $\mathbf{b}$ may be calculated by considering the sequence of emitting states $\mathsf{s}_1, \mathsf{s}_2, \ldots, \mathsf{s}_p$ during a realization of the PHMM. The PHMM starts in the state $\mathsf{s}_0 =$ START: to obtain a proper probability distribution over all (ordered) pairs of sequences, we must require that after $\mathbf{a}$ and $\mathbf{b}$ have been produced, the PHMM enters state $\mathsf{s}_{p+1} =$ END.

We will derive the probability that $\mathbf{a}$ and $\mathbf{b}$ are the complete sequences emitted by the PHMM in three steps: first, by considering the conditional probabilities of adding one or two symbols to existing strings during the realization; next, by considering the probability that strings $\mathbf{a}$ and $\mathbf{b}$ are produced at some point during a realization (but may be extended); and finally by considering the probability that the PHMM stops after producing $\mathbf{a}$ and $\mathbf{b}$.

Consider a particular time point during the realization. Given that the sequences emitted so far are $\mathbf{c}$ and $\mathbf{d}$, and given that the current state is s, let $p(\mathbf{c}a, \mathbf{d}, \mathsf{t} \mid \mathbf{c}, \mathbf{d}, \mathsf{s})$

denote the probability that the next state will be a singly emitting state t, that t will emit the symbol a which is appended to $\mathbf{c}$. Let $p(\mathbf{c}, \mathbf{d}b, \mathsf{t} \mid \mathbf{c}, \mathbf{d}, \mathsf{s})$ and $p(\mathbf{c}a, \mathbf{d}b, \mathsf{t} \mid \mathbf{c}, \mathbf{d}, \mathsf{s})$ be defined similarly.
Then we have:

$$\begin{aligned} p(\mathbf{c}a, \mathbf{d}, \mathsf{t} \mid \mathbf{c}, \mathbf{d}, \mathsf{s}) &= T(\mathsf{s}, \mathsf{t})E(\mathsf{t}, a, \cdot) \\ p(\mathbf{c}, \mathbf{d}b, \mathsf{t} \mid \mathbf{c}, \mathbf{d}, \mathsf{s}) &= T(\mathsf{s}, \mathsf{t})E(\mathsf{t}, \cdot, b) \\ p(\mathbf{c}a, \mathbf{d}b, \mathsf{t} \mid \mathbf{c}, \mathbf{d}, \mathsf{s}) &= T(\mathsf{s}, \mathsf{t})E(\mathsf{t}, a, b) \end{aligned} \tag{3.16}$$

Let $p(\mathbf{c}a, \mathbf{d}b, \mathsf{t})$ be the probability that at some point during a realization, the sequences produced so far are $\mathbf{c}a$ and $\mathbf{d}b$, and the current state (after the last symbol emission) is t. It follows that

$$\begin{aligned} p(\mathbf{c}a, \mathbf{d}b, \mathsf{t}) = &\sum_{\mathsf{s}} p(\mathbf{c}, \mathbf{d}b, \mathsf{s})T(\mathsf{s}, \mathsf{t})E(\mathsf{t}, a, \cdot) \\ &+ \sum_{\mathsf{s}} p(\mathbf{c}a, \mathbf{d}, \mathsf{s})T(\mathsf{s}, \mathsf{t})E(\mathsf{t}, \cdot, b) \\ &+ \sum_{\mathsf{s}} p(\mathbf{c}, \mathbf{d}, \mathsf{s})T(\mathsf{s}, \mathsf{t})E(\mathsf{t}, a, b) \end{aligned} \tag{3.17}$$

Using this equation, it is possible to build up the probabilities $p(\mathbf{c}, \mathbf{d}, \mathsf{s})$ for all prefixes of $\mathbf{a}$, $\mathbf{b}$ by starting with null strings and adding the symbols one at a time, storing all probabilities computed to use in subsequent stages. Finally, the probability that $\mathbf{a}$ and $\mathbf{b}$ are exactly the strings produced by the PHMM is

$$p(\mathbf{a}, \mathbf{b}) = \sum_{\mathsf{s}} p(\mathbf{a}, \mathbf{b}, \mathsf{s})T(\mathsf{s}, \text{END}) \tag{3.18}$$

The number of computations required to calculate $p(\mathbf{a}, \mathbf{b})$ is $O(|\mathbf{a}||\mathbf{b}||\mathcal{S}|)$. For further details, consult [Durbin et al., 1998].

3.5 Conditionally Symmetrically Independent PHMMs

The state diagram of a useful CSI PHMM is shown in Figure 3.5 below.

The state AB emits matching, or nearly matching symbols for both sequences; the states A and B emit *insertions*, parts of one sequence that are not parts of the other. ϵ, δ, and γ are all small probabilities. The most frequently taken state-transitions are drawn with thicker arrows. The PHMM starts in START, and then typically repeatedly cycles through AB. Occasionally it will reach the state A or B, and then generate an insertion of several symbols, before going back to AB. Eventually, the state END will be reached, and the process will stop.

This PHMM is useful even though it only has three states that emit symbols, which is the minimum number for a non-trivial PHMM. The joint distribution defined by this PHMM gives high probabilities to sequences that match along large parts of their lengths, where "match" means that pairs of corresponding symbols are generated by the state AB.

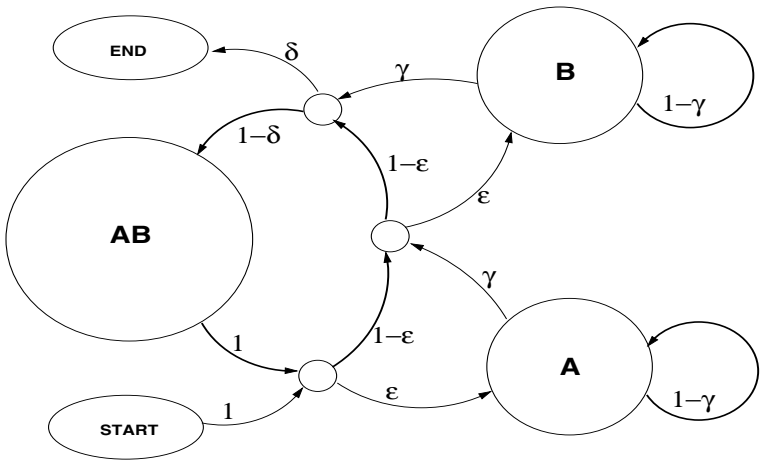


Figure 3.1 A CSI pair HMM for Matching

To state sufficient conditions for a PHMM $\mathcal{H}$ to be CSI requires some definitions. Let T^{AB} be the transition probabilities restricted to $\mathcal{S}^{AB}$. That is, for $\mathsf{s},\mathsf{t} \in \mathcal{S}^{AB}$, let $T^{AB}(\mathsf{s},\mathsf{t})$ be the probability that, starting from s, the next state in $\mathcal{S}^{AB}$ reached is t.

Let $\mathsf{A}^{\uparrow}(\mathsf{s},\mathsf{t})$ be the random variable denoting the possibly empty subsequence of states in $\mathcal{S}^{A}$ that the process passes through, given that the process starts in state $\mathsf{s} \in \mathcal{S}^{AB}$, and given that state t is the next state in $\mathcal{S}^{AB}$ reached. Let $\mathsf{B}^{\uparrow}(\mathsf{s},\mathsf{t})$ be a random variable defined similarly.

Definition 3.2
A PHMM $\mathcal{H}$ has the *independent insertion property* if, for all $\mathsf{s},\mathsf{t} \in \mathcal{S}^{AB}$, $\mathsf{A}^{\uparrow}(\mathsf{s},\mathsf{t})$ and $\mathsf{B}^{\uparrow}(\mathsf{s},\mathsf{t})$ are independent.

Proposition 3.3
Let $\mathcal{H}$ be a PHMM such that:

1. The joint distribution over sequences induced by $\mathcal{H}$ is unchanged if $\mathcal{S}^{A}$ is relabelled as $\mathcal{S}^{B}$ and $\mathcal{S}^{B}$ as $\mathcal{S}^{A}$.
2. For all states $s \in \mathcal{S}^{AB}$, the symbol-emission joint p.d. over $\mathcal{B} \times \mathcal{B}$ is CSI.
3. $\mathcal{H}$ has the independent insertion property.

main result

Then the joint p.d. induced by $\mathcal{H}$ over pairs of sequences of symbols is CSI.

Proof The proof is in two stages. It is shown first that any PHMM that satisfies condition 2 may be transformed into an equivalent PHMM in which all states in $\mathcal{S}^{AB}$ have symmetric independent joint emission distributions. Next, it is shown that the probability of a realization may be factored so that sequences A and B are independent given the subsequence of states from $\mathcal{S}^{AB}$ that occurs in the realization. The result follows.

From condition 2, it follows for each $\mathsf{s} \in \mathcal{S}^{AB}$, the symbol-emission p.d. is a mixture of symmetric independent distributions. It is possible to construct an equivalent PHMM to $\mathcal{H}$ in which all states in $\mathcal{S}^{AB}$ have symmetric independent emission distributions, by replacing each state in $\mathcal{S}^{AB}$ with a network of states.

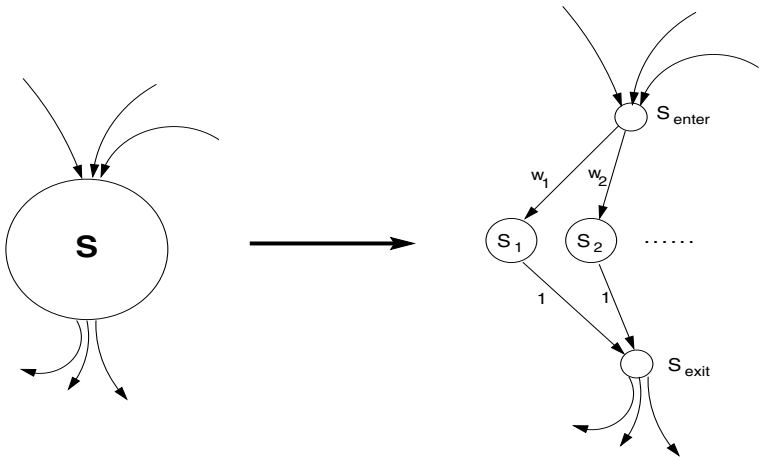


Figure 3.2 Re-writing a doubly emitting state as a mixture of atomic states

As shown in Figure 3.2, the state s can be decomposed into a non-emitting entry state $\mathsf{s}_{\mathsf{enter}}$, a set of alternative *atomic* doubly emitting states $\mathsf{s}_1, \mathsf{s}_2, \ldots$ and an exit state $\mathsf{s}_{\mathsf{exit}}$. The number of atomic states may be finite or countably infinite: note that even if there are infinitely many atomic states, the entire PHMM is still, by construction, equivalent to finite PHMM in the sense that it generates an identical joint p.d. over symbol sequences.

For each state t for which a transition to s is possible, the transition occurs to $\mathsf{s}_{\mathsf{enter}}$ with the same probability. From $\mathsf{s}_{\mathsf{enter}}$, there is a transition to one of the atomic states $\mathsf{s}_1, \mathsf{s}_2, \ldots$, the transition to s_i having probability w_i. From s_i there is a transition with probability 1 to $\mathsf{s}_{\mathsf{exit}}$, and from $\mathsf{s}_{\mathsf{exit}}$ the transition probabilities are the same as from s. The distribution of symbols emitted by the substituted network of states consisting of $\mathsf{s}_{\mathsf{enter}}$, $\mathsf{s}_1, \mathsf{s}_2, \ldots$, and $\mathsf{s}_{\mathsf{exit}}$ is exactly the same as the distribution of symbols emitted by s.

The point of this substitution is that all of the doubly emitting states $\mathsf{s}_1, \mathsf{s}_2, \ldots$ now emit pairs of *independent* symbols. From now on, therefore, we may assume that all states in $\mathcal{S}^{AB}$ emit pairs of *independent* symbols.

■ Let ω be a realization of the PHMM $\mathcal{H}$. Let ω contain $n+1$ states from $\mathcal{S}^{AB}$. Let the sequence of states from $\mathcal{S}^{AB}$ be $\mathsf{c} = \langle \mathsf{c}_0, \ldots, \mathsf{c}_n \rangle$, with $\mathsf{c}_0 = \text{START}$ and $\mathsf{c}_n = \text{END}$.

■ Let $\mathsf{a}_i^{\uparrow}$ be the possibly empty sequence of states from $\mathcal{S}^A$ that occur between c_{i-1} and c_i in ω, and let $\mathsf{b}_i^{\uparrow}$ be defined similarly.

■ Let $\mathbf{a}(\mathsf{c}_i)$ denote the symbol in sequence $\mathbf{a}$ emitted by the state c_i, and let $\mathbf{b}(\mathsf{c}_i)$ be defined similarly.

■ Let $\mathsf{a}^\uparrow = \left\langle \mathsf{a}_0^\uparrow, \ldots, \mathsf{a}_n^\uparrow \right\rangle$ and let $\mathsf{b}^\uparrow = \left\langle \mathsf{b}_0^\uparrow, \ldots, \mathsf{a}_n^\uparrow \right\rangle$ be the complete sequences of insertions of states in $\mathcal{S}^A$ and $\mathcal{S}^B$ respectively.

We seek to show that $p(\mathbf{a}, \mathbf{b} \mid \mathsf{c}) = p(\mathbf{a} \mid \mathsf{c})\ p(\mathbf{b} \mid \mathsf{c})$. Now, from the independent insertion property,

$$p(\mathsf{a}_i^\uparrow, \mathsf{b}_i^\uparrow \mid \mathsf{c}_{i-1}, \mathsf{c}_i) = p(\mathsf{a}_i^\uparrow \mid \mathsf{c}_{i-1}, \mathsf{c}_i)\ p(\mathsf{b}_i^\uparrow \mid \mathsf{c}_{i-1}, \mathsf{c}_i) \tag{3.19}$$

for $1 \leq i \leq n$, so that

$$\begin{aligned} p(\mathsf{a}^\uparrow, \mathsf{b}^\uparrow \mid \mathsf{c}) &= \prod_{i=1}^{n} p(\mathsf{a}_i^\uparrow, \mathsf{b}_i^\uparrow \mid \mathsf{c}_{i-1}, \mathsf{c}_i) \\ &= p(\mathsf{a}^\uparrow \mid \mathsf{c}) p(\mathsf{b}^\uparrow \mid \mathsf{c}) \end{aligned} \tag{3.20}$$

As each c_i is an atomic state with an independent emission distribution,

$$p(\mathbf{a}(\mathsf{c}_i), \mathbf{b}(\mathsf{c}_i) \mid \mathsf{c}_i) = p(\mathbf{a}(\mathsf{c}_i) \mid \mathsf{c}_i)\ p(\mathbf{b}(\mathsf{c}_i) \mid \mathsf{c}_i) \tag{3.21}$$

for $1 \leq i \leq n$, and since states in $\mathcal{S}^A$ do not affect symbols in $\mathbf{b}$, and vice versa, it follows from (3.21) that

$$p(\mathbf{a}, \mathbf{b} \mid \mathsf{a}^\uparrow, \mathsf{b}^\uparrow, \mathsf{c}) = p(\mathbf{a} \mid \mathsf{a}^\uparrow, \mathsf{c})\ p(\mathbf{b} \mid \mathsf{b}^\uparrow, \mathsf{c}) \tag{3.22}$$

Hence

$$p(\mathbf{a}, \mathbf{b} \mid \mathsf{c}) = \sum_{\mathsf{a}^\uparrow, \mathsf{b}^\uparrow} p(\mathbf{a}, \mathbf{b} \mid \mathsf{a}^\uparrow, \mathsf{b}^\uparrow, \mathsf{c})\ p(\mathsf{a}^\uparrow, \mathsf{b}^\uparrow \mid \mathsf{c}) \tag{3.23}$$

$$= \sum_{\mathsf{a}^\uparrow, \mathsf{b}^\uparrow} \left(p(\mathbf{a} \mid \mathsf{a}^\uparrow, \mathsf{c}) p(\mathsf{a}^\uparrow \mid \mathsf{c})\right) \left(p(\mathbf{b} \mid \mathsf{b}^\uparrow, \mathsf{c}) p(\mathsf{b}^\uparrow \mid \mathsf{c})\right) \tag{3.24}$$

$$= p(\mathbf{a} \mid \mathsf{c})\ p(\mathbf{b} \mid \mathsf{c}) \tag{3.25}$$

where (3.24) follows from (3.22), (3.20) and rearrangement of terms. ■

This proof shows that a natural and currently used matching function for sequences can be explicitly represented as a scalar product. The feature space has one dimension for each possible sequence of atomic doubly emitting states c; the number of such c for which the mapping $\phi(\mathbf{a})$ is non-zero is in general exponential in the length of the symbol sequence $\mathbf{a}$.

3.6 Conclusion

A natural, currently used class of match-scores for sequences have been shown to be representable as scalar products in a high-dimensional space. It follows that these match-scores can be used in dual formulations of linear statistical methods, and also that the match-scores may be used to locate sequences in a Euclidean space.

We are investigating possible applications and extensions of this approach for bio-sequence analysis and speech recognition.

Acknowledgments

We have recently learned that Tommi Jaakkola (tommi@ai.mit.edu) with David Haussler invented and studied CSI kernels and CSI PHMM kernels before we did; his work is currently unpublished, but part of it is described in a recent technical report [Haussler, 1999], which gives an extensive discussion of kernels on discrete structures. Our work is independent of theirs.

Volodya Vovk and John Shawe-Taylor made helpful suggestions. This work started partly as the result of email correspondence with Saira Mian and Inna Dubchak.

4 Natural Regularization from Generative Models

Nuria Oliver
Media Arts and Sciences Laboratory
MIT, 20 Ames Street, E15-384C
Cambridge, MA 02139, USA
nuria@media.mit.edu
http://www.media.mit.edu/~nuria/

Bernhard Schölkopf
Microsoft Research Limited
St. George House, 1 Guildhall Street
Cambridge CB2 3NH, UK
bsc@microsoft.com
http://www.research.microsoft.com/~bsc/

Alexander J. Smola
Department of Engineering
Australian National University
Canberra 0200 ACT, Australia
Alex.Smola@anu.edu.au
http://spigot.anu.edu.au/~smola/

Recently, Jaakkola and Haussler proposed the so-called Fisher kernel to construct discriminative kernel techniques by using generative models. We provide a regularization-theoretic analysis of this approach and extend the set of kernels to a class of natural kernels, all based on generative models with density $p(\mathbf{x}|\theta)$ like the original Fisher kernel. This allows us to incorporate distribution dependent smoothness criteria in a general way.

As a result of this analyis we show that the Fisher kernel corresponds to a $L_2(p)$ norm regularization. Moreover it allows us to derive explicit representations of the eigensystem of the kernel, give an analysis of the spectrum of the integral operator, and give experimental evidence that this may be used for model selection purposes.

4.1 Introduction

Learning Theory using discriminative and generative models has enjoyed significant progress over the last decade. Generative techniques such as HMMs, dynamic graphical models, or mixtures of experts have provided a principled framework for dealing with missing and incomplete data, uncertainty or variable length sequences. On the other hand, discriminative models like SV Machines [Boser et al., 1992] and other kernel methods (Gaussian Processes [Williams, 1998], Regularization Networks [Girosi et al., 1995], etc.) have become standard tools of applied machine learning technology, leading to record benchmark results in a variety of domains. However, until recently, these two strands have been largely separated.

A promising approach to combine the strengths of both worlds by designing kernels inspired by generative models was made in the work of Jaakkola and Haussler [1999b,a] (cf. Watkins' Chapter 3 for an alternative approach). They propose the use of a so-called Fisher kernel to give a "natural" similarity measure taking into account an underlying probability distribution.

Since defining a kernel function automatically implies assumptions about metric relations between the examples, they argue that these relations should be defined directly from a generative probability model $p(\mathbf{x}|\theta)$, where θ are the parameters of the model. Their choice is justified from two perspectives: that of improving the discriminative power of the model and from an attempt to find a 'natural' comparison between examples induced by the generative model.

While this is quite an abstract concept, it would be desirable to obtain a deeper understanding of the regularization properties of the resulting kernel. In other words, it would be instructive to see which sort of functions such a kernel favours, which degrees of smoothness are chosen, or how categorical data is treated. Many of these properties can be seen by deriving the regularization operator (with the associated prior) [Smola et al., 1998a] to which such a kernel corresponds to.

The chapter is structured as follows. In Section 4.2 we introduce tools from information geometry and define a class of *natural kernels* to which also the two kernels proposed by Jaakkola and Haussler [1999b] belong. A regularization theoretic analysis of natural kernels follows in Section 4.3. In particular we show that the so-called Fisher kernel corresponds to a prior distribution over the functions $f(\theta)$ taking the form $p(f) \propto \exp\left(-\frac{1}{2}\|f\|_p^2\right)$, where $\|\cdot\|_p^2$ is the norm of the $L_2(p)$ space of functions square integrable wrt. the measure corresponding to $p(x|\theta)$, i.e., the usual norm weighted by the underlying generative model. Finally, in Section 4.4 we derive the decomposition of natural kernels into their eigensystem which allows to describe the image of input space in feature space. The shape of the latter has consequences for the generalization behavior of the associated kernel method (cf., e.g., [Williamson et al., 1998]). Section 4.5 concludes the chapter with some experiments and a discussion.

4.2 Natural Kernels

Conventional SV kernels like the ones introduced in Section 1.3.2 by Eq. (1.63), (1.72) or (1.73) ignore knowledge of the underlying distribution of the data $p(x)$ which could be provided by a generative model or additional information about the problem at hand. Instead, a general requirement of smoothness is imposed [Girosi, 1998, Smola et al., 1998b]. This may not always be desirable, e.g., in the case of categorical data (attributes such as english, german, spanish, ...) and sometimes one may want to enforce a higher degree of smoothness where data is sparse, and less smoothness where data is abundant. Both issues will be addressed in the following.

To introduce a class of kernels derived from generative models, we need to introduce basic concepts of information geometry. Consider a family of generative models $p(\mathbf{x}|\theta)$ (i.e., probability measures) smoothly parametrized by θ. These models form a manifold (also called statistical manifold) in the space of all probability measures. The key idea introduced by Jaakkola and Haussler [1999b] is to exploit the geometric structure on this manifold to obtain an (induced) metric for the training patterns $\mathbf{x}_i$. Rather than dealing with $p(\mathbf{x}|\theta)$ directly one uses the log-likelihood instead, i.e., $l(\mathbf{x}, \theta) := \ln p(\mathbf{x}|\theta)$.

score map

- The derivative map of $l(\mathbf{x}|\theta)$ is usually called the **score map** $U_\theta : \mathcal{X} \to \mathbb{R}^r$ with

$$U_\theta(\mathbf{x}) := (\partial_{\theta^1} l(\mathbf{x}, \theta), \ldots, \partial_{\theta^r} l(\mathbf{x}, \theta)) = \nabla_\theta l(\mathbf{x}, \theta) = \nabla_\theta \ln p(\mathbf{x}|\theta), \tag{4.1}$$

whose coordinates are taken as a 'natural' basis of tangent vectors. Note that θ is the coordinate system for any parametrization of the probability density $p(\mathbf{x}|\theta)$. For example, if $p(\mathbf{x}|\theta)$ is a normal distribution, one possible parametrization would be $\theta = (\mu, \sigma)$, where μ is the mean vector and σ is the covariance matrix of the Gaussian. The basis given by the score map represents the direction in which the value of the ith coordinate increases while the others are fixed.

- Since the manifold of $\ln p(\mathbf{x}|\theta)$ is Riemannian, there is an inner product defined in its tangent space T_p whose metric tensor is given by the inverse of the **Fisher information matrix**

Fisher information matrix

$$I(p) := E_p\left[U_\theta(\mathbf{x}) U_\theta(\mathbf{x})^\top\right] \text{ i.e., } I_{ij}(p) = E_p\left[\partial_{\theta^i} \ln p(\mathbf{x}|\theta) \partial_{\theta^j} \ln p(\mathbf{x}|\theta)\right]. \tag{4.2}$$

Here E_p denotes the expectation with respect to the density p.

- This metric is called the **Fisher information metric** and induces a 'natural' distance in the manifold. It can be used to measure the difference in the generative process between a pair of examples $\mathbf{x}_i$ and $\mathbf{x}_j$ via the score map $U_\theta(\mathbf{x})$ and I^{-1}.

Note that the metric tensor, i.e., I_p^{-1}, depends on p and therefore on the parametrization θ. This is different to the conventional Euclidean metric on $\mathbb{R}^n$ where the metric tensor is simply the identity matrix. For the purposes of calculation it is often easier to compute I_{ij} as the Hessian of the scores:

Hessian of the scores

$$I(p) = -E_p\left(\nabla_\theta \nabla_\theta^\top \ln p(\mathbf{x}|\theta)\right) \text{ with } I_{ij}(p) = -E_p\left(\partial_{\theta_i} \partial_{\theta_j} \ln p(\mathbf{x}|\theta)\right) \tag{4.3}$$

In summary, what we need is a family of probability measures for which the log-likelihood $l(\mathbf{x}, \theta) = \ln p(\mathbf{x}|\theta)$ is a differentiable map.

Definition 4.1 Natural Kernel
Denote by M a positive definite matrix and by $U_\theta(x)$ the score map defined above. Then the corresponding natural kernel is given by

$$k_M^{\text{nat}}(\mathbf{x}, \mathbf{x}') := U_\theta(\mathbf{x})^\top M^{-1} U_\theta(\mathbf{x}') = \nabla_\theta \ln p(\mathbf{x}|\theta)^\top M^{-1} \nabla_\theta \ln p(\mathbf{x}'|\theta) \tag{4.4}$$

In particular, if $M = I$, hence k_I^{nat}, the (4.4) reduces to the **Fisher kernel** [Jaakkola and Haussler, 1999b]. Moreover if $M = \mathbf{1}$ one obtains a kernel we will call the **plain kernel** which is often used for convenience if I is too difficult to compute.[1]

In the next section, we will give a regularization theoretic analysis of the class of natural kernels, hence in particular of k_I^{nat} and k_1^{nat}. This answers the question to which type of smoothness (or rather 'simplicity') the kernels proposed in [Jaakkola and Haussler, 1999b] correspond to.

4.3 The Natural Regularization Operator

Let us briefly recall the theory of Section 1.3.3. In SV machines one minimizes a regularized risk functional (1.80) where the complexity term can be written as $\frac{\lambda}{2}\|w\|^2$ in feature space notation, or as $\frac{\lambda}{2}\|Pf\|^2$ when considering the functions in input space directly. In particular, the connection between kernels k, feature spaces F and regularization operators P is given by (1.82) which is repeated below for the sake of convenience.

$$k(\mathbf{x}_i, \mathbf{x}_j) = ((Pk)(\mathbf{x}_i, .) \cdot (Pk)(\mathbf{x}_j, .)) \,. \tag{4.5}$$

It states that if k is a *Greens* function of P^*P, minimizing $\|w\|$ in feature space is equivalent to minimizing the regularized risk functional given by $\|Pf\|^2$.

To analyze the properties of natural kernels k_I^{nat}, we exploit this connection between kernels and regularization operators by finding the operator P_M^{nat} such that (4.5) holds. To this end, we need to specify a dot product in (4.5). Note that this is part of the choice of the class of regularization operators that we are looking at — in particular, it is a choice of the dot product space that P maps into. We opt for the dot product in $L_2(p)$ space, i.e.,

$$\langle f, g \rangle := \int f(\mathbf{x}) g(\mathbf{x}) p(\mathbf{x}|\theta) d\mathbf{x} \tag{4.6}$$

1. For the sake of correctness one would have to write $k^{\text{nat}}_{M,p(\mathbf{x},\cdot)}$ rather than k_M^{nat} since k also depends on the generative model and the parameter θ chosen by some other procedure such as density estimation. Moreover note that rather than requiring M to be positive definite, semidefiniteness would be sufficient. However, then, we would have to replace M^{-1} by the pseudoinverse and the subsequent reasoning would be significantly more cumbersome.

since this will lead to a simple form of the corresponding regularization operators. Other measures would also have been possible, leading to different formal representations of P.

Proposition 4.2 Regularization Operators for Natural Kernels
Given a positive definite matrix M, a generative model $p(\mathbf{x}|\theta)$, and a corresponding natural kernel $k_M^{\text{nat}}(\mathbf{x}, \mathbf{x}')$, P_M^{nat} is an equivalent regularization operator if it satisfies the following condition:

$$M = \int \left[P_M^{\text{nat}} \nabla_\theta \ln p(\mathbf{z}|\theta)\right] \left[P_M^{\text{nat}} \nabla_\theta \ln p(\mathbf{z}|\theta)\right]^\top p(\mathbf{z}|\theta) d\mathbf{z} \tag{4.7}$$

Proof Substituting (4.4) into (4.5) yields

$$k_M^{\text{nat}}(\mathbf{x}, \mathbf{x}') \stackrel{\text{by def}}{=} \nabla_\theta \ln p(\mathbf{x}|\theta)^\top M^{-1} \nabla_\theta \ln p(\mathbf{x}'|\theta) \tag{4.8}$$

$$\stackrel{(4.5)}{=} \left\langle P_M^{\text{nat}} k_M^{\text{nat}}(\mathbf{x}, \mathbf{z}), P_M^{\text{nat}} k_M^{\text{nat}}(\mathbf{x}', \mathbf{z}) \right\rangle \tag{4.9}$$

$$= \int \nabla_\theta \ln p(\mathbf{x}|\theta)^\top M^{-1} \left[P_M^{\text{nat}} \nabla_\theta \ln p(\mathbf{z}|\theta)\right] \times \left[P_M^{\text{nat}} \nabla_\theta \ln p(\mathbf{z}|\theta)^\top\right] M^{-1} \nabla_\theta \ln p(\mathbf{x}'|\theta) p(\mathbf{z}|\theta) d\mathbf{z} \tag{4.10}$$

Note that P_M^{nat} acts on p as a function of $\mathbf{z}$ only — the terms in $\mathbf{x}$ and $\mathbf{x}'$ are not affected which is why we may collect them outside. Thus the necessary condition (4.7) ensures that the rhs (4.9) equals (4.10) which completes the proof. ■

Let us consider the two special cases proposed by Jaakkola and Haussler [1999b].

Fisher regularization operator

Corollary 4.3 Fisher Kernel
The Fisher Kernel ($M = I$) induced by a generative probability model with density p corresponds to a regularizer equal to the squared $L_2(p)$-norm of the estimated function. Therefore the regularization term is given by

$$\|Pf\|^2 = \|f\|^2_{L_2(p)}. \tag{4.11}$$

This can be seen by substituting in $P_I^{\text{nat}} = \mathbf{1}$ into the rhs of (4.7) which yields the definition of the Fisher information matrix.

To get an intuition about what this regularizer does, let us spell it out explicitly. The solution of SV regression using the Fisher kernel has the form $f(\mathbf{x}) = \sum_{i=1}^m \alpha_i k_I^{\text{nat}}(\mathbf{x}, \mathbf{x}_i)$, where the $\mathbf{x}_i$ are the SVs, and $\boldsymbol{\alpha}$ is the solution of the SV programming problem. Applied to this function, we obtain

$$\|f(\theta)\|^2_{L_2(p)} = \int |f(\mathbf{x})|^2 p(\mathbf{x}|\theta) d\mathbf{x} \tag{4.12}$$

$$= \int \left(\sum\nolimits_i \alpha_i \nabla_\theta \ln p(\mathbf{x}|\theta) I^{-1} \nabla_\theta \ln p(\mathbf{x}_i|\theta)\right)^2 p(\mathbf{x}|\theta) d\mathbf{x}.$$

To understand this term, first recall that what we actually minimize is the regularized risk $R_{\text{reg}}[f]$, the sum of (4.12) and the empirical risk given by the normalized negative log likelihood. The regularization term (4.12) prevents overfitting by favoring solutions with smaller $\nabla_\theta \ln p(\mathbf{x}|\theta)$. Consequently, the regularizer will favor

the solution which is more stable (flat). Figure 4.1 illustrates this effect.

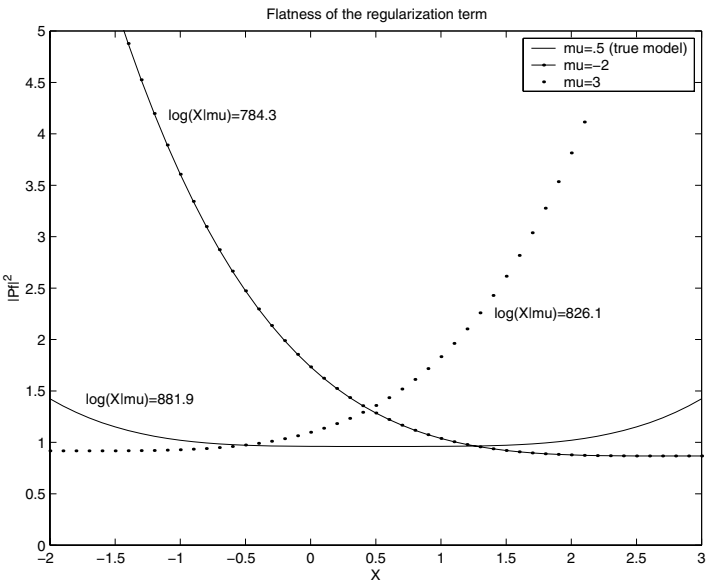


Figure 4.1 Flatness of the natural regularizer for a Gaussian generative pdf $\sim \mathcal{N}(0.5, 3)$, $\theta = (\mathbf{0.5}, \mathbf{3})$. Let us assume we are given two parameter vectors θ_1 and θ_2 which both lead to the same high likelihood. In this case, the regularizer will pick the parameter vector with the property that *perturbing* it will (on average) lead to a smaller change in the log likelihood, for in that case $\nabla_\theta \ln p(\mathbf{x}|\theta)$ will be smaller. Consequently, the regularizer will favor the solution which is more stable (flat).

Note, however, that the validity of this intuitive explanation is somewhat limited since some effects can compensate each other as the α_i come with different signs. Finally, we remark that the regularization operator of the conformal transformation [Amari and Wu, 1999] of the Fisher kernel k_I^{nat} into $\sqrt{p(\mathbf{x}|\theta)}\sqrt{p(\mathbf{x}'|\theta)}k_I^{\text{nat}}(\mathbf{x}, \mathbf{x}')$ is the identity map in L_2 space.

In practice, Jaakkola and Haussler [1999b] often use $M = \mathbf{1}$. In this case, Proposition 4.2 specializes to the following result.

Corollary 4.4 Plain Kernel
The regularization operator associated with the plain kernel $k_{\mathbf{1}}^{\text{nat}}$ is the gradient operator $\nabla_{\mathbf{x}}$ in the case where $p(\mathbf{x}|\theta)$ belongs to the exponential family of densities, i.e., $\ln p(\mathbf{x}|\theta) = \theta \cdot \mathbf{x} - \pi(\mathbf{x}) + c_0$.

Proof We substitute $\ln p(\mathbf{x}|\theta)$ into the condition (4.7). This yields

$$\int \left[\nabla_{\mathbf{z}} \nabla_\theta \ln p(\mathbf{z}|\theta)\right]^\top \left[\nabla_{\mathbf{z}} \nabla_\theta \ln p(\mathbf{z}|\theta)\right] p(\mathbf{z}|\theta) d\mathbf{z}$$

$$= \int \left[\nabla_{\mathbf{z}}(\mathbf{z} - \nabla_\theta \pi(\mathbf{x}))\right]^\top \left[\nabla_{\mathbf{z}}(\mathbf{z} - \nabla_\theta \pi(\mathbf{x}))\right] p(\mathbf{z}|\theta) d\mathbf{z} = \mathbf{1}. \tag{4.13}$$

since the terms depending only on $\mathbf{z}$ vanish after application ∇_θ. ■

This means that the regularization term can be written as (note $\nabla_{\mathbf{x}} f(\mathbf{x}$ is a vector)

$$\|Pf\|^2 = \|\nabla_{\mathbf{x}} f(\mathbf{x})\|_p^2 = \int \|\nabla_{\mathbf{x}} f(\mathbf{x})\|^2 p(\mathbf{x}|\theta) d\mathbf{x} \tag{4.14}$$

thus favouring smooth functions via flatness in the first derivative. Often one is facing the opposite problem of identifying a kernel k_M^{nat} from its corresponding regularization operator P. This can be solved by evaluating (4.7) for the appropriate class of operators. A possible choice would be Radon-Nikodym derivatives, i.e., $p^{-1}(\mathbf{x})\nabla_{\mathbf{x}}$ [Canu and Elisseeff, 1999] or powers thereof. In this regard (4.7) is particularly useful, since methods such as the probability integral transform which can be used to obtain Greens functions for Radon-Nikodym operators in $\mathbb{R}$ by mapping $\mathbb{R}$ into $[0,1]$ with density 1, cannot be extended to $\mathbb{R}^n$.

4.4 The Feature Map of Natural Kernel

Given a regularization operator P with an expansion P^*P into a discrete eigensystem (λ_n, ψ_n), where λ are the eigenvalues and ψ the eigenvectors, and given a kernel k with

$$k(\mathbf{x}_i, \mathbf{x}_j) := \sum_n \frac{d_n}{\lambda_n} \psi_n(\mathbf{x}_i)\psi_n(\mathbf{x}_j) \tag{4.15}$$

where $d_n \in 0, 1$ for all m, and $\sum_n \frac{d_n}{\lambda_n}$ convergent. Then k satisfies the self-consistency property stated in equation (4.5) [Smola et al., 1998b]. For the purpose of designing a kernel with regularization properties given by P, eq. (4.15) is a constructive version of Mercer's Theorem (Th. 1.16).

The eigenvalues of the Gram Matrix of the training set are used to bound the generalization error of a margin classifier [Schölkopf et al., 1999]. By linear algebra we may explicitly construct such an expansion (4.15).

Proposition 4.5 Map into Feature Space
Denote by I the Fisher information matrix, by M the kernel matrix, and by s_i, Λ_i the eigensystem of $M^{-\frac{1}{2}} I M^{-\frac{1}{2}}$. The kernel $k_M^{\text{nat}}(\mathbf{x}, \mathbf{x}')$ can be decomposed into an eigensystem

$$\psi_i(\mathbf{x}) = \frac{1}{\sqrt{\Lambda_i}} s_i^\top M^{-\frac{1}{2}} \nabla_\theta \ln p(\mathbf{x}|\theta) \text{ and } \lambda_i = \Lambda_i. \tag{4.16}$$

Note that if $M = I$ we have $\lambda_i = \Lambda_i = 1$.

Proof It can be seen immediately that (4.15) is satisfied. This follows from the fact that s_i is an orthonormal basis, ($\mathbf{1} = \sum_i s_i s_i^\top$) and the definition of k_M^{nat}. The terms depending on Λ_i cancel out mutually.

The second part (orthonormality of ψ_i) can be seen as follows.

$$\langle \psi_i, \psi_j \rangle \tag{4.17}$$

$$= \int \left(\frac{1}{\sqrt{\Lambda_i}} s_i^\top M^{-\frac{1}{2}} \nabla_\theta \ln p(\mathbf{x}|\theta) \right) \left(\frac{1}{\sqrt{\Lambda_j}} \nabla_\theta^\top \ln p(\mathbf{x}|\theta) M^{-\frac{1}{2}} s_j \right) p(\mathbf{x}|\theta) d\mathbf{x}$$

$$= \frac{1}{\sqrt{\Lambda_i \Lambda_j}} s_i^\top M^{-\frac{1}{2}} I M^{-\frac{1}{2}} s_j = \delta_{ij} \tag{4.18}$$

This completes the proof. ■

unit eigenvalues

The eigenvalues λ_i^I of k_I^{nat} are all 1, reflecting the fact that the matrix I whitens the scores $\nabla_\theta \ln(p(x|\theta))$. It also can be seen from $P_I = \mathbf{1}$ that (4.16) becomes $\psi_i(x) = \frac{1}{\sqrt{\lambda_i^I}} s_i \cdot \nabla_\theta \ln(p(x|\theta))$, $1 \leq i \leq r$.

What are the consequences of the fact that all eigenvalues are equal? Standard VC dimension bounds [Vapnik, 1995] state that the capacity of a linear classifier or regression algorithm is essentially given by $R^2 \cdot \Lambda^2$. Here, R is the radius of the smallest sphere containing the data (in feature space), and Λ is the maximal allowed length of the weight vector. Recently, it has been shown that both the spectrum of an associated integral operator [Williamson et al., 1998] and the spectrum of the Gram matrix $k((\mathbf{x}_i, \mathbf{x}_j))_{ij}$ [Schölkopf et al., 1999] can be used to formulate generalization error bounds. This was done by exploiting the fact that since $C := \sup_j \|\psi_j\|_{L_\infty}$ exists, (4.16) implies that $|\Phi_i(x)| = \sqrt{\lambda_i}|\psi_i(\mathbf{x})| \leq \sqrt{\lambda_i} C$, i.e., the mapped data live in some parallelepiped whose sidelengths are given by the square roots of the eigenvalues. New bounds improved upon the generic VC dimension bounds by taking into account this fact: due to the decay of the eigenvalues, the mapped data are not distributed isotropically. Therefore capturing the shape of the mapped data only by the radius of a sphere should be a rather rough approximation. On the other hand, taking into account the rate of decay of the eigenvalues allows one to formulate kernel-dependent bounds which are much more accurate than the standard VC-bounds.

In our case all λ_i are 1, therefore $|\Phi_i(\mathbf{x})| = |\psi_i(\mathbf{x})|$. Hence the upper bound simply states that the mapped data is contained in some box with equal sidelengths (hypercube). Moreover, the $L_2(p)$ normalization of the eigenfunctions ψ_i means that $\int \psi_i(x)^2 p(x|\theta)\ dx = 1$. Therefore, the squared averaged size of the feature map's ith coordinate is independent of i, implying that the the mapped data have the same range in all directions. This isotropy of the Fisher kernels suggests that the standard 'isotropic' VC bounds should be fairly precise in this case.

4.5 Experiments

The flat eigenspectrum of the Fisher kernel suggests a way of comparing different models: we compute the Gram matrix for a set of $\mathcal{K}$ models $p(\mathbf{x}|\theta^j)$ with $j = 1 \ldots \mathcal{K}$. In the case of the true model, we expect $\lambda_i = 1$ for all i. Therefore one might select the model j such that its spectrum is the flattest. As a sanity check for the theory developed, Figure 4.5 illustrates the selection of the sufficient statistics (μ, σ) of a one-dimensional normal pdf $p(\mathbf{x}|\theta) = \mathcal{N}(\mu, \sigma)$ with 10 training data points sampled from $\mathcal{N}(0.5, 3)$. We computed the eigendecomposition of the empirical Gram matrices, using the Fisher kernels of a set of different models. The figure contains the error bar plots of the ratio of its 2 largest eigenvalues (note that in this case the parameter space is two-dimensional). The minimum corresponds to the model to be selected.

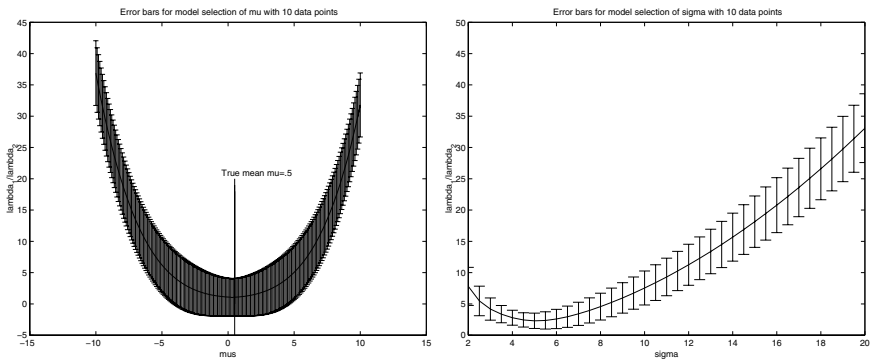


Figure 4.2 Model selection using the ratio of the two largest eigenvalues of the empirical Gram Matrix. Right: selecting the standard deviation. Left: selecting the mean

4.6 Discussion

In this chapter we provided a regularization-theoretic analysis of a class of SV kernels — called natural kernels — based on generative models with density $p(\mathbf{x}|\theta)$, such as the Fisher kernel. In particular, we have shown that the latter corresponds to a regularization operator (prior) penalizing the $L_2(p)$-norm of the estimated function. Comparing this result to the regularization-theoretic analysis of SV kernels [Smola et al., 1998a], where common SV kernels such as the Gaussian have been shown to correspond to a sum over differential operators of different orders, the question arises whether it is possible to find a modified natural kernel which uses

higher order derivatives in the regularization term, such as

$$\|Pf\|^2 = \sum_{n=0}^{\infty} c_n \|\nabla^n f\|^2_{L_2(p)} . \tag{4.19}$$

Second, we derived the feature map corresponding to natural kernels. It turned out that the Fisher natural kernel corresponding to a r-parameter generative model maps the input data into a r-dimensional feature space where the data are distributed isotropically (in the sense that the covariance matrix is the identity). This reflects the fact that all parameters are considered equally important, and that the Fisher kernel is invariant with respect to parameter rescaling; it automatically scales feature space in a principled way. Our analysis provides some understanding for the impressive empirical results obtained using the Fisher kernel.

Acknowledgments

Thanks to Shun-Ichi Amari, André Elisseeff, Klaus-Robert Müller, and Si Wu for helpful discussions. Parts of this work were done while AS, BS, and NO were at GMD FIRST and University of Madison. It was supported by the DFG (grants Ja 379/52,71,91 and Sm 62/1-1).

5 Probabilities for SV Machines

John C. Platt
Microsoft Research
1 Microsoft Way, Redmond, WA 98052
jplatt@microsoft.com
http://research.microsoft.com/~jplatt

The output of a classifier should be a calibrated posterior probability to enable post-processing. Standard SVMs do not provide such probabilities. One method to create probabilities is to directly train a kernel classifier with a logit link function and a regularized maximum likelihood score. However, training with a maximum likelihood score will produce non-sparse kernel machines. Instead, we train an SVM, then train the parameters of an additional sigmoid function to map the SVM outputs into probabilities. This chapter compares classification error rate and likelihood scores for an SVM plus sigmoid versus a kernel method trained with a regularized likelihood error function. These methods are tested on three data-mining-style data sets. The SVM+sigmoid yields probabilities of comparable quality to the regularized maximum likelihood kernel method, while still retaining the sparseness of the SVM.

5.1 Introduction

Constructing a classifier to produce a posterior probability $P(\text{class}|\text{input})$ is very useful in practical recognition situations. For example, a posterior probability allows decisions that can use a utility model [Duda and Hart, 1973]. Posterior probabilities are also required when a classifier is making a small part of an overall decision, and the classification outputs must be combined for the overall decision. An example of this combination is using a Viterbi search or HMM to combine recognition results from phoneme recognizers into word recognition [Bourlard and Morgan, 1990]. Even in the simple case of a multi-category classifier, choosing the category based on maximal posterior probability over all classes is the Bayes optimal decision for the equal loss case.

However, Support Vector Machines [Vapnik, 1998] (SVMs) produce an uncalibrated value that is not a probability. Let the unthresholded output of an SVM be

$$f(\mathbf{x}) = h(\mathbf{x}) + b, \tag{5.1}$$

where

$$h(\mathbf{x}) = \sum_i y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) \tag{5.2}$$

lies in a Reproducing Kernel Hilbert Space (RKHS) $\mathcal{F}$ induced by a kernel k [Wahba, 1999b]. Training an SVM minimizes an error function that penalizes an approximation to the training misclassification rate plus a term that penalizes the norm of h in the RKHS:

SVM Error

$$C \sum_i (1 - y_i f_i)_+ + \frac{1}{2}||h||_{\mathcal{F}}, \tag{5.3}$$

where $f_i = f(\mathbf{x}_i)$. Minimizing this error function will also minimize a bound on the test misclassification rate [Vapnik, 1998], which is also a desirable goal. An additional advantage of this error function is that minimizing it will produce a sparse machine where only a subset of possible kernels are used in the final machine.

One method of producing probabilistic outputs from a kernel machine was proposed by Wahba [1992, 1999b]. Wahba used a logistic link function,

$$P(\text{class}|\text{input}) = P(y = 1|\mathbf{x}) = p(\mathbf{x}) = \frac{1}{1 + \exp(-f(\mathbf{x}))}, \tag{5.4}$$

where f is defined as above, and then proposed minimizing a negative log multinomial likelihood plus a term that penalizes the norm in an RKHS:

Maximum Likelihood Error

$$-\frac{1}{m} \sum_i \left(\frac{y_i + 1}{2} \log(p_i) + \frac{1 - y_i}{2} \log(1 - p_i) \right) + \lambda ||h||^2_{\mathcal{F}}, \tag{5.5}$$

where $p_i = p(\mathbf{x}_i)$. The output $p(\mathbf{x})$ of such a machine will be a posterior probability. Minimizing this error function will not directly produce a sparse machine, but a modification to this method can produce sparse kernel machines [Wahba, 1999a].

This chapter presents modifications to SVMs which yield posterior probabilities, while still maintaining their sparseness. First, the chapter reviews recent work in modifying SVMs to produce probabilities. Second, it describes a method for fitting a sigmoid that maps SVM outputs to posterior probabilities. Finally, the SVM plus sigmoid combination is compared to a regularized likelihood fit using the same kernel on three different data-mining-style data sets.

5.1.1 Recent Work

Vapnik

[Vapnik, 1998, sec. 11.11] suggests a method for mapping the output of SVMs to probabilities by decomposing the feature space $\mathcal{F}$ into a direction orthogonal to the separating hyperplane, and all of the $N - 1$ other dimensions of the feature

space. The direction orthogonal to the separating hyperplane is parameterized by t (a scaled version of $f(\mathbf{x})$), while all of the other directions are parameterized by a vector $\mathbf{u}$. In full generality, the posterior probability depends on both t and $\mathbf{u}$: $P(y = 1|t, \mathbf{u})$. Vapnik proposes fitting this probability with a sum of cosine terms:

$$P(y = 1|t, \mathbf{u}) = a_0(\mathbf{u}) + \sum_{n=1}^{N} a_n(\mathbf{u}) \cos(nt). \tag{5.6}$$

The coefficients of the cosine expansion will minimize a regularized functional [Vapnik, 1998, eqn. 7.93], which can be converted into a linear equation for the a_n that depends on the value of $\mathbf{u}$ for the current input being evaluated.

Preliminary results for this method, shown in [Vapnik, 1998, Figure 11.8], are promising. However, there are some limitations that are overcome by the method of this chapter. For example, the Vapnik method requires a solution of a linear system for every evaluation of the SVM. The method of this chapter does not require a linear system solver call per evaluation because it averages the $P(y = 1|f)$ over all $\mathbf{u}$. The price of this efficiency is that dependencies of $P(y = 1|f)$ on $\mathbf{u}$ cannot be modeled. Another interesting feature of the Vapnik method is that the sum of the cosine terms is not constrained to lie between 0 and 1, and is not constrained to be monotonic in f. See, for example, [Vapnik, 1998, Figure 11.8]. There is a very strong prior for considering the probability $P(y = 1|f)$ to be monotonic in f, since the SVM is trained to separate most or all of the positive examples from the negative examples.

Hastie & Tibshirani

Another method for fitting probabilities to the output of an SVM is to fit Gaussians to the class-conditional densities $p(f|y = 1)$ and $p(f|y = -1)$. This was first proposed by Hastie and Tibshirani [1998], where a single tied variance is estimated for both Gaussians. The posterior probability rule $P(y = 1|f)$ is thus a sigmoid, whose slope is determined by the tied variance. Hastie and Tibshirani [1998] then adjust the bias of the sigmoid so that the point $P(y = 1|f) = 0.5$ occurs at $f = 0$. This sigmoid is monotonic, but the single parameter derived from the variances may not accurately model the true posterior probability.

Gaussian Fit

One can also use a a more flexible version of the Gaussian fit to $p(f|y = \pm 1)$. The mean and the variance for each Gaussian is determined from a data set. Bayes' rule can be used to compute the posterior probability via:

$$P(y = 1|f) = \frac{p(f|y = 1)P(y = 1)}{\sum_{i=-1,1} p(f|y = i)P(y = i)}, \tag{5.7}$$

where $P(y = i)$ are prior probabilities that can be computed from the training set.[1] In this formulation, the posterior is an analytic function of f with form:

$$P(y = 1|f) = \frac{1}{1 + \exp(af^2 + bf + c)}. \tag{5.8}$$

1. This model for SVM output probabilities was independently proposed and used for speaker identification in a talk by C. J. C. Burges at the 1998 NIPS SVM workshop.

There are two issues with this model of SVM outputs. First, the posterior estimate derived from the two-Gaussian approximation violates the strong monotonic prior mentioned above: the function in (5.8) is non-monotonic. Second, the assumption of Gaussian class-conditional densities is often violated (see Figure 5.1).

5.2 Fitting a Sigmoid After the SVM

5.2.1 Motivation

Instead of estimating the class-conditional densities $p(f|y)$, we use a parametric model to fit the posterior $P(y = 1|f)$ directly. The parameters of the model are adapted to give the best probability outputs.

The form of the parametric model can be inspired by looking at empirical data. Figure 5.1 shows a plot of the class-conditional densities $p(f|y = \pm 1)$ for a linear SVM trained on a version of the UCI Adult data set (see [Platt, 1999]). The plot shows histograms of the densities (with bins 0.1 wide), derived from threefold cross-validation. These densities are very far away from Gaussian. There are discontinuities in the derivatives of both densities at both the positive margin $f = 1$ and the negative margin $f = -1$. These discontinuities are not surprising, considering that the cost function (5.3) also has discontinuities at the margins. Theory to explain the form of these class-conditional densities is currently under development.

The class-conditional densities between the margins are apparently exponential. Bayes' rule (5.7) on two exponentials suggests using a parametric form of a sigmoid:

Sigmoid

$$P(y = 1|f) = \frac{1}{1 + \exp(Af + B)}. \tag{5.9}$$

This sigmoid model is equivalent to assuming that the output of the SVM is proportional to the log odds of a positive example. This sigmoid model is different from that proposed in [Hastie and Tibshirani, 1998] because it has two parameters trained discriminatively, rather than one parameter estimated from a tied variance.

The sigmoid fit works well, as can be seen in Figure 5.2. The data points in Figure 5.2 are derived by using Bayes' rule on the histogram estimates of the class-conditional densities in Figure 5.1. For a linear SVM trained on the Adult data set [Platt, 1999], the sigmoid fits the non-parametric estimate extremely well, even beyond the margins. On the other sets and other kernels described in this chapter, the sigmoid fits reasonably well, with a small amount of bias beyond the margins. The non-parametric model of posterior probability for handwritten digits shown in [Vapnik, 1998, Figure 11.8] is also very close to a sigmoid. Therefore, the sigmoid posterior model seems to be close to the true model.

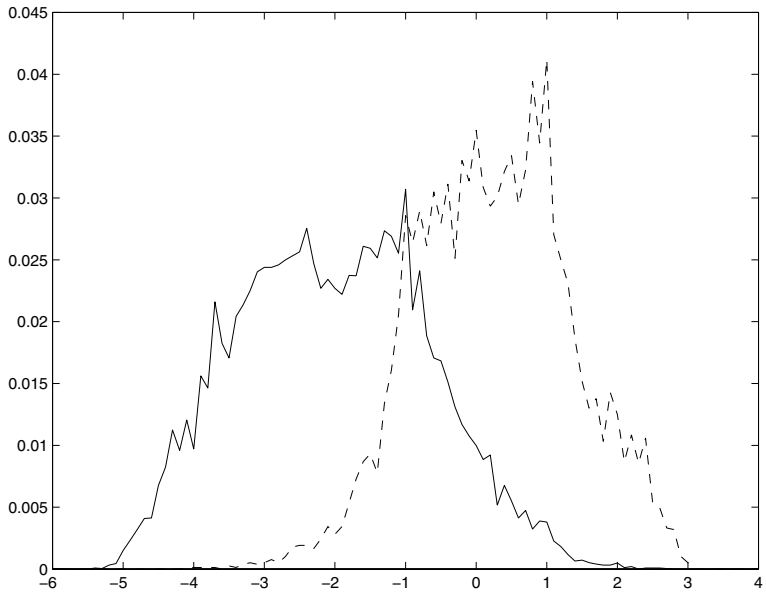


Figure 5.1 The histograms for $p(f|y = \pm 1)$ for a linear SVM trained on the Adult data set. The solid line is $p(f|y = -1)$, while the dashed line is $p(f|y = 1)$. Notice that these histograms are not Gaussian.

One can also view the sigmoid function as a linearization (in log-odds space) of the posterior in (5.8). As long as $A < 0$, the monotonicity of (5.9) is assured. Even if, in some cases, the class-conditional densities are close to Gaussian, the sigmoid fit is still appropriate and valuable.

5.2.2 Fitting the Sigmoid

The parameters A and B of (5.9) are fit using maximum likelihood estimation from a training set (f_i, y_i). First, let us define a new training set (f_i, t_i), where the t_i are target probabilities defined as:

$$t_i = \frac{y_i + 1}{2}. \tag{5.10}$$

Sigmoid Error Function

The parameters A and B are found by minimizing the negative log likelihood of the training data, which is a cross-entropy error function:

$$\min -\sum_i t_i \log(p_i) + (1 - t_i) \log(1 - p_i), \tag{5.11}$$

where

$$p_i = \frac{1}{1 + \exp(Af_i + B)}. \tag{5.12}$$

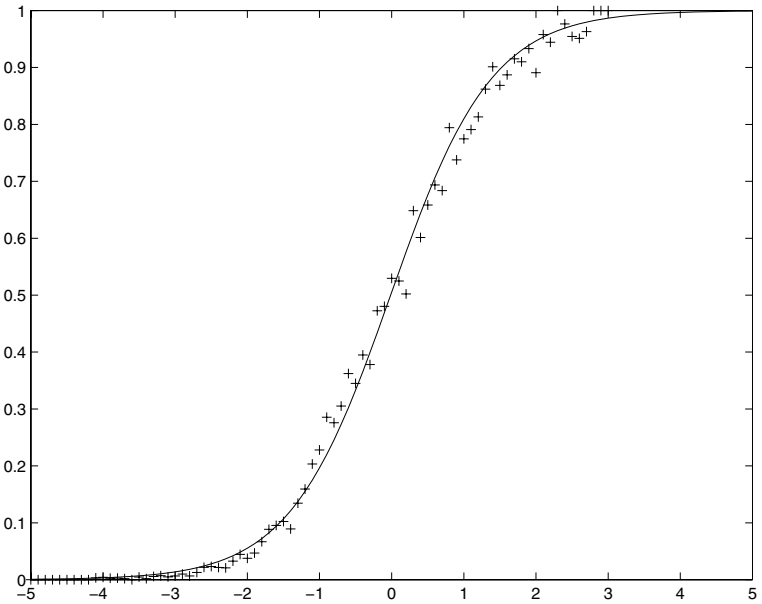


Figure 5.2 The fit of the sigmoid to the data for a linear SVM on the Adult data set (as in Figure 5.1). Each plus mark is the posterior probability computed for all examples falling into a bin of width 0.1. The solid line is the best-fit sigmoid to the posterior, using the algorithm described in this chapter.

The minimization in (5.11) is a two-parameter minimization. Hence, it can be performed using any number of optimization algorithms. For robustness, the experiments in this chapter were performed using a model-trust minimization algorithm [Gill et al., 1981], whose pseudo-code is shown in Appendix 5.5.

Two issues arise in the optimization of (5.11): the choice of the sigmoid training set (f_i, y_i), and the method to avoid over-fitting this set.

Training Set Choice

The easiest training set to use is simply the same training examples used to fit the SVM. That is, $f_i = f(x_i)$, where x_i is the ith training example. However, the training of the SVM causes the SVM outputs f_i to be a biased estimate of the distribution of f out of sample. For examples at the margin, the f_i are forced to have absolute value exactly 1, which certainly will not be a common value for test examples. The training examples that fail the margin $(1 - y_i f_i > 0)$ are also subtly biased, since the f_i are pushed towards the margin by the corresponding α_i. Only the f_i that are beyond the margin are substantially unbiased.

For linear SVMs, the bias introduced by training usually is not severe. In almost all cases, a maximum of $N + 1$ support vectors will lie on the margin (for an input dimensionality of N), which is usually a small fraction of the training set. Also, for many real-world problems that use linear SVMs, optimal performance is reached for small C, which causes the bias on the margin failures to become small. Therefore, for linear SVMs, it often possible to simply fit the sigmoid on the training set.

For non-linear SVMs, the support vectors often form a substantial fraction of the entire data set, especially when the Bayes error rate for the problem is high [Vapnik, 1998]. Through empirical experiments, fitting a sigmoid to the training set of non-linear SVMs sometimes leads to disastrously biased fits. Therefore, we must form an unbiased training set of the output of the SVM f_i.

One method for forming an unbiased training set is to approximate leave-one-out estimates of f_i, as described in Chapter 15. However, this either requires the solution of a linear system for every data point in the training set, or a re-run of an SVM solver at every data point, which can be computationally expensive.

Hold-Out Set

There are two computationally inexpensive methods for deriving an unbiased training set: generating a hold-out set and cross-validation. To use a hold out set, a fraction of the training set (typically 30%) is not used to train the SVM, but is used to train the sigmoid. This same hold-out set can be used to estimate other parameters of the system, such as kernel choice, kernel parameters, and C. Once A, B, and all of the other system parameters are determined from the hold out set, the main SVM can be re-trained on the entire training set. If SVM training scales roughly quadratically with training set size [Platt, 1999, Joachims, 1999], then the hold-out set will be only 1.5 times slower than simply training on the entire data set. Because determining the system parameters is often unavoidable, determining A and B from the hold-out set may not incur extra computation with this method.

Cross Validation

Cross-validation is an even better method than a hold-out set for estimating the parameters A and B [Kearns, 1997]. In three-fold cross-validation, the training set is split into three parts. Each of three SVMs are trained on permutations of two out of three parts, and the f_i are evaluated on the remaining third. The union of all three sets of f_i can form the training set of the sigmoid (and also can be used to adjust the SVM system parameters). Cross-validation produces larger sigmoid training sets than the hold-out method, and hence gives a lower variance estimate for A and B. Three-fold cross-validation takes approximately 2.2 times as long as training a single SVM on an entire training set. All of the results in this chapter are presented using three-fold cross-validation.

Even with cross-validated unbiased training data, the sigmoid can still be overfit. For example, in the Reuters data set [Dumais, 1998, Joachims, 1998], some of the categories have very few positive examples which are linearly separable from all of the negative examples. Fitting a sigmoid for these SVMs with maximum likelihood will simply drive the parameter A to a very large negative number, even if the positive examples are reweighted. There can be an infinite number of solutions with infinitely steep sigmoids when the validation set is perfectly separable. Therefore, we must regularize to prevent overfitting to a small number of examples.

Regularization

Regularization requires either a prior model for the parameter space (A, B), or a prior model for a distribution of out-of-sample data. One can imagine using a Gaussian or Laplacian prior on A. However, there is always one free parameter in the prior distribution (e.g., the variance). This free parameter can be set using cross-validation or Bayesian hyperparameter inference [MacKay, 1992], but these methods add complexity to the code.

A simpler method is to create a model of out-of-sample data. One model is to assume that the out-of-sample data is simply the training data perturbed with Gaussian noise. This is the model behind Parzen windows [Duda and Hart, 1973, Vapnik, 1998]. However, this model still has a free parameter.

Non-Binary Targets

The sigmoid fit in this chapter uses a different out-of-sample model: out-of-sample data is modelled with the same empirical density as the sigmoid training data, but with a finite probability of opposite label. In other words, when a positive example is observed at a value f_i, we do not use $t_i = 1$, but assume that there is a finite chance of opposite label at the same f_i in the out-of-sample data. Therefore, a value of $t_i = 1 - \epsilon_+$ will be used, for some ϵ_+. Similarly, a negative example will use a target value of $t_i = \epsilon_-$. Using a non-binary target does not require any modification to the maximum likelihood optimization code. Because (5.11) is simply the Kullback-Liebler divergence between f_i and t_i, the function is still well-behaved, even for non-binary t_i.

The probability of correct label can be derived using Bayes' rule. Let us choose a uniform uninformative prior over probabilities of correct label. Now, let us observe N_+ positive examples. The MAP estimate for the target probability of positive examples is

$$t_+ = \frac{N_+ + 1}{N_+ + 2}. \tag{5.13}$$

Similarly, if there are N_- negative examples, then the MAP estimate for the target probability of negative examples is

$$t_- = \frac{1}{N_- + 2}. \tag{5.14}$$

These targets are used instead of $\{0, 1\}$ for all of the data in the sigmoid fit.

These non-binary targets value are Bayes-motivated, unlike traditional non-binary targets for neural networks [Rumelhart et al., 1986a]. Furthermore, the non-binary targets will converge to $\{0, 1\}$ when the training set size approaches infinity, which recovers the maximum likelihood sigmoid fit.

The pseudo-code in Appendix 5.5 shows the optimization using the Bayesian targets.

5.3 Empirical Tests

There are at least two experiments to determine the real-world performance of the SVM+sigmoid combination. First, the SVM+sigmoid can be compared to a plain SVM for misclassification rate. Assuming equal loss for Type I and Type II errors, the optimal threshold for the SVM+sigmoid is $P(y = 1|f) = 0.5$, while the optimal threshold for the SVM is $f = 0$. This first experiment checks to see if the 0 threshold is optimal for SVMs.

The second experiment is to compare the SVM+sigmoid with a kernel machine trained to explicitly maximize a log multinomial likelihood. For the linear kernel

case, this is equivalent to comparing a linear SVM to regularized logistic regression. The purpose of the second experiment is to check the quality of probability estimates by the SVM+sigmoid hybrid combination, and see if the error function (5.3) causes fewer misclassifications than (5.5). Three different classification tasks were used.

Task	Training Set Size	Testing Set Size	C	Number of Inputs	Number of SVMs
Reuters Linear	9603	3299	0.08	300	118
Adult Linear	32562	16282	0.05	123	1
Adult Quadratic	1605	16282	0.3	123	1
Web Linear	49749	21489	1.0	300	1
Web Quadratic	2477	21489	10.0	300	1

Table 5.1 Experimental Parameters

The first task is determining the category of a Reuters news article [Dumais, 1998, Joachims, 1998]. The second task is the UCI Adult benchmark of estimating the income of a household given census form data [Blake et al., 1998], where the input vectors are quantized [Platt, 1999]. The third task is determining the category of a web page given key words in the page [Platt, 1999]. The Reuters task is solved using a linear SVM, while the Adult and Web tasks are solved with both linear and quadratic SVMs. The parameters of the training are shown in Table 5.1. The regularization terms are set separately for each algorithm, via performance on a hold-out set. The C value shown in Table 5.1 is for the SVM+sigmoid. The sigmoid parameters are estimated using three-fold cross-validation. The quadratic kernel for the Adult task is

$$k(\mathbf{x}_i, \mathbf{x}_j) = \left(\frac{\mathbf{x}_i \cdot \mathbf{x}_j + 1}{14}\right)^2, \tag{5.15}$$

while the quadratric kernel for the Web task is

$$k(\mathbf{x}_i, \mathbf{x}_j) = \left(\frac{\mathbf{x}_i \cdot \mathbf{x}_j + 1}{12}\right)^2; \tag{5.16}$$

The constants 12 and 14 are taken from the average over each data set of the dot product of an example with itself. This normalization keeps the kernel function in a reasonable range.

Table 5.2 shows the results of these experiments. The table lists the number of errors for a raw SVM, an SVM+sigmoid, and a regularized likelihood kernel method. It also lists the negative log likelihood of the test set for SVM+sigmoid and for the regularized likelihood kernel method. McNemar's test [Dietterich, 1998] was used to find statistically significant differences in classification error rate, while the Wilcoxson signed rank test [Mosteller and Rourke, 1973] is used to find significant differences in the log likelihood. Both of these tests examine the results of a pair of algorithms on every example in the test set. In Table 5.2, underlined entries

are pairwise statistically significantly better than all non-underlined entries, while not statistically significantly better than any other underlined entry. A significance threshold of $p = 0.05$ is used.

Task	Raw SVM Number of Errors	SVM + Sigmoid Number of Errors	Regularized Likelihood Number of Errors	SVM + Sigmoid $-\log(p)$ Score	Regularized Likelihood $-\log(p)$ Score
Reuters Linear	1043	963	1060	3249	3301
Adult Linear	2441	2442	2434	5323	5288
Adult Quadratic	2626	2554	2610	5772	5827
Web Linear	260	265	248	1121	958
Web Quadratic	444	452	507	1767	2163

Table 5.2 Experimental Results

5.3.1 Discussion

Three interesting results were observed from these experiments. First, adding a sigmoid sometimes improves the error rate of a raw SVM: a zero threshold is not necessarily Bayes optimal. For the Reuters Linear and Adult Quadratic tasks, the sigmoid threshold was significantly better than the standard zero threshold. For both of these tasks, the ratio of the priors $P(y = -1)/P(y = 1)$ is far from one, which will tend to push the Bayes optimal threshold away from zero. For example, on the Adult Quadratic task, the threshold $P(y = 1|f) = 0.5$ corresponds to a threshold of $f = -0.1722$, which is simply a more optimal threshold than zero. The VC bounds on the generalization error [Vapnik, 1998] do not guarantee that the zero threshold is Bayes optimal.

The second interesting result is that adding the sigmoid produces probabilities of roughly comparable quality to the regularized likelihood kernel method. For three of the five tasks, the regularized likelihood yields significantly better probabilities. For the Web Quadratic task, the SVM+sigmoid has a better overall log likelihood, but the Wilcoxon rank test prefers the regularized likelihood kernel method because more data points are more accurate with the latter method.

The third interesting result is that neither the SVM+sigmoid nor the regularized likelihood kernel machine is a completely dominant method for either error rate or log likelihood. The SVM+sigmoid makes fewer errors than the regularized likelihood kernel method for three out of five tasks, while the regularized likelihood method makes fewer errors for one out of five tasks. This result is somewhat surprising: the SVM kernel machine is trained to minimize error rate, while the regularized likelihood is trained to maximize log likelihood. These experiments indicate that, when all other factors (e.g., kernel choice) are held constant, the difference in performance between (5.3) and (5.5) is hard to predict *a priori*.

Finally, it is interesting to note that there are other kernel methods that produce sparse machines without relying on an RKHS. One such class of methods penalize the ℓ_1 norm of the function h in (5.3), rather than the RKHS norm [Mangasarian, 1965, Chen et al., 1999] (see, for example, Chapter 8). Fitting a sigmoid after fitting these sparse kernel machines may, in future work, yield reasonable estimates of probabilities.

5.4 Conclusions

This chapter presents a method for extracting probabilities $P(\text{class}|\text{input})$ from SVM outputs, which is useful for classification post-processing. The method leaves the SVM error function (5.3) unchanged. Instead, it adds a trainable post-processing step which is trained with regularized binomial maximum likelihood. A two parameter sigmoid is chosen as the post-processing, since it matches the posterior that is empirically observed. Finally, the SVM+sigmoid combination is compared to a raw SVM and a kernel method entirely trained with regularized maximum likelihood. The SVM+sigmoid combination preserves the sparseness of the SVM while producing probabilities that are of comparable quality to the regularized likelihood kernel method.

Acknowledgments

I would like to thank Chris Bishop for valuable advice during the writing of the chapter.

5.5 Appendix: Pseudo-code for the Sigmoid Training

This appendix shows the pseudo-code for the training is shown below. The algorithm is a model-trust algorithm, based on the Levenberg-Marquardt algorithm [Press et al., 1992].

```
Input parameters:
  out = array of SVM outputs
  target = array of booleans: is ith example a positive example?
  prior0 = number of negative examples
  prior1 = number of positive examples
  len = number of training examples
Outputs:
  A, B = parameters of sigmoid

A = 0
B = log((prior0+1)/(prior1+1))
hiTarget = (prior1+1)/(prior1+2)
loTarget = 1/(prior0+2)
lambda = 1e-3
olderr = 1e300
pp = temp array to store current estimate of probability of examples
set all pp array elements to (prior1+1)/(prior0+prior1+2)
count = 0
for it = 1 to 100 {
  a = 0, b = 0, c  = 0, d = 0, e = 0
// First, compute Hessian & gradient of error function
// with respect to A & B
  for i = 1 to len {
     if (target[i])
        t = hiTarget
     else
        t = loTarget
     d1 = pp[i]-t
     d2 = pp[i]*(1-pp[i])
     a += out[i]*out[i]*d2
     b += d2
     c += out[i]*d2
     d += out[i]*d1
     e += d1
  }
// If gradient is really tiny, then stop
  if (abs(d) < 1e-9 && abs(e) < 1e-9)
     break
```

```
  oldA = A
  oldB = B
  err = 0
// Loop until goodness of fit increases
  while (1) {
    det = (a+lambda)*(b+lambda)-c*c
    if (det == 0) {  // if determinant of Hessian is zero,
                     // increase stabilizer
      lambda *= 10
      continue
    }
    A = oldA + ((b+lambda)*d-c*e)/det
    B = oldB + ((a+lambda)*e-c*d)/det
    // Now, compute the goodness of fit
    err = 0;
    for i = 1 to len {
      p = 1/(1+exp(out[i]*A+B))
      pp[i] = p
      // At this step, make sure log(0) returns -200
      err -= t*log(p)+(1-t)*log(1-p)
    }
    if (err < olderr*(1+1e-7)) {
      lambda *= 0.1
      break
    }
    // error did not decrease: increase stabilizer by factor of 10
    // & try again
    lambda *= 10
    if (lambda >= 1e6) // something is broken. Give up
      break
  }
  diff = err-olderr
  scale = 0.5*(err+olderr+1)
  if (diff > -1e-3*scale && diff < 1e-7*scale)
    count++
  else
    count = 0
  olderr = err
  if (count == 3)
    break
}
```

6 Maximal Margin Perceptron

Adam Kowalczyk
Telstra Research Laboratories
770 Blackburn Road
Clayton, Vic. 3168
Australia
a.kowalczyk@trl.oz.au

A local learning rule (a modification of the classical perceptron) is presented and shown to converge to the "optimal hyperplane" separating data points with a maximal separation margin ρ. We show that after at most $\frac{2D^2}{\epsilon^2\rho^2}\ln\frac{D}{2\rho}$ updates a (finite or infinite) data set will be separated with a margin larger than $(1-\epsilon)\rho$, for any $0<\epsilon<1$, where D is the diameter of the data set. The results are extended to the kernel case and then the soft margin case with quadratic penalty. Some initial experimental results including a comparison with six other algorithms for iterative generation of support vector machines are also presented.

6.1 Introduction

Training a support vector machine requires the solution of a quadratic optimization task [Cortes and Vapnik, 1995, Vapnik, 1998]. In the case of large data sets (several thousand data points) this requires deployment of complex, subtle and sometimes difficult to implement procedures. A significant effort has been devoted recently to the development of simplified solutions of this quadratic optimization task. One direction here is centered on splitting the solution of the soft-margin problem into a series of smaller size subtasks [Cortes and Vapnik, 1995, Osuna et al., 1997a, Vapnik, 1998, Kaufman, 1998, Joachims, 1999, Platt, 1999]. Those methods rely on batch processing since the selection of a subtask (an active set) requires an examination for the whole training set. The extreme case here is the SMO algorithm reducing the solution of soft margin case to a series of "two point" subproblems [Platt, 1999].

link to Adatron

Another approach is represented by recently proposed extensions of the Adatron algorithm to the kernel machine case (*kernel adatron*) [Frieß et al., 1998, Frieß, 1999]. These algorithms are suitable for both on-line and batch implementations, and in this respect they are close to results in this chapter. The kernel adatron is based on previous research in the statistical physics of learning, e.g., [Krauth and Mézard, 1987, Anlauf and Biehl, 1989]. A number of interesting theoretical results on Adatron have been obtained, including estimates of convergence rates to optimal solutions based on replica calculations [Opper, 1989, Watkin et al., 1993], although, at this stage they are applicable to special cases of the kernel adatron only.[1]

The approach developed in this chapter is based on solving the original (primal) problem rather than the dual problem via satisfying KKT conditions as used in SMO. The underlying idea is to generate a solution by approximating the closest points between two convex polytopes (formed by convex hulls of the data points separated according to the class to which they belong). A similar approach was also independently taken by Keerthi et al. [1999] (this was brought to our attention by reviewers of the first draft of this chapter). Their paper is built upon earlier work in control theory, where a number of algorithms for determining the closest point of a convex polytope from the origin were developed, e.g., [Gilbert, 1966, Michell et al., 1974]. However, apart from a similar general direction, the final results and algorithms in this chapter and in [Keerthi et al., 1999] are different. To be precise, in our algorithms we concentrate on single point updates selected to minimize the total number of updates in both on-line and batch modes of learning, whereas Keerthi et al. [1999] use updates based on multiple points and consider only batch learning (following approach of Michell et al. [1974]). Furthermore, we provide proofs of convergence rate and computational complexity, whereas Keerthi et al. [1999] stop short of this by showing that approximation is possible in finite time. As both papers investigate different heuristics for selection of updates, the bounds on resulting convergence rates differ (for batch learning). In Section 6.6 we compare some algorithms introduced in this chapter with those benchmarked by Keerthi et al. [1999], by running ours on the same set of benchmarks. Interestingly, the link to papers of Gilbert [1966] and Michell et al. [1974] shows that this chapter provides novel results of interest in control theory. We leave more detailed discussion to Section 6.7.

In this chapter we first show that the case of separable data (hard margin) can be solved "on-line," i.e., with an evaluation of one data point at a time and making a necessary correction. This procedure will ultimately converge to the "optimal hyperplane" separating data with the maximal margin. As all operations required are linear, the generalization to the kernel case is standard. In this case, in batch learning mode, it is also advantageous to minimize the number of updates by using a greedy search for updates giving maximal "gain." The computational overhead

1. Those restrictions include the homogeneous (no bias) case, some selected learning rates, specific input vectors with ± 1 entries, etc.

due to such a search is quite small, especially in the case of high dimensional input spaces. Such an algorithm is discussed in Section 6.3.1.

In this chapter we also concentrate on proofs of convergence to the optimal solutions and estimates of the rate of convergence. The proofs presented here have straightforward geometrical motivations and hold for the "hard margin" case in full generality (no restrictions on the distribution of training samples, the results are valid for a finite set of iterations rather than the thermodynamic limit, etc.) In Section 6.5 the results are extended to the "soft margin" case with (quadratic penalty).

chapter organization

The chapter is organized as follows. The basic algorithms are introduced in Section 6.3 together with convergence proofs. Both on-line and batch, greedy search based, algorithms are considered. Section 6.4 describes an extension of the basic algorithms to the kernel case. This is further advanced to the soft margin case with quadratic penalty in Section 6.5, where a convergence theorem for an algorithm with a global greedy search is given. A kernel version of a greedy search algorithm has been evaluated experimentally on NIST handwritten digit benchmark with some results of experiments presented in Section 6.6.1. In Section 6.6.2 and the Appendix we present a comparison of this algorithm with six other iterative procedures for generation of support vector machines which have been previously benchmarked by Keerthi et al. [1999]. The results are discussed in Section 6.7. The Appendix gives some details of benchmark tests.

6.2 Basic Approximation Steps

We are given a training sequence $(\mathbf{x}_i, y_i) \in \mathbb{R}^N \times \{-1, 1\}$, $i = 1, \ldots, m$. We introduce the notation $I^{(+1)} := \{i\ ;\ y_i = 1\}$, $I^{(-1)} := \{i\ ;\ y_i = -1\}$, , $X^{(+1)} := \{\mathbf{x}_i\ ;\ y_i = 1\}$ and $X^{(-1)} := \{\mathbf{x}_i\ ;\ y_i = -1\}$ for the indices and data points with positive and negative labels, respectively.

The data is called *linearly separable* if there exists a linear functional (shortly *a functional*)

$$\pi(\mathbf{x}) = \pi_{\mathbf{w},b}(\mathbf{x}) := \mathbf{w} \cdot \mathbf{x} + b \qquad (\forall \mathbf{x} \in \mathbb{R}^N), \tag{6.1}$$

such that

$$y_i \pi(\mathbf{x}_i) > 0 \qquad (i = 1, \ldots, m). \tag{6.2}$$

It is well known that if the data is linearly separable, then $(\mathbf{w}, b) \in \mathbb{R}^N \times \mathbb{R}$ as above can be found by the classical perceptron algorithm in a finite number of iterations. We recall, that the *margin* of the linear functional (6.1) is given by (see also (1.12))

$$\rho(\pi) := \min_{i=1,\ldots,m} \frac{y_i \pi(\mathbf{x}_i)}{\|\mathbf{w}\|} = \min_{i=1,\ldots,m} \frac{y_i(\mathbf{w} \cdot \mathbf{x}_i + b)}{\|\mathbf{w}\|}. \tag{6.3}$$

Note that π uniquely determines $\mathbf{w}$ and b. Obviously, $\rho(\pi)$ is the largest number d such that $y_i \pi(\mathbf{x}_i) \geq d\|\mathbf{w}\|$ for $i = 1, \ldots, m$, and the data is separable by π iff

$\rho(\pi) > 0$. A functional π_* with *the maximal margin*

$$\rho := \max_{(\mathbf{w},b)\in(\mathbb{R}^N-0)\times\mathbb{R}} \rho(\pi_{\mathbf{w},b}) = \rho(\pi_*) \tag{6.4}$$

is called *optimal* and the hyperplane ${\pi_*}^{-1}(0)\subset\mathbb{R}^N$ is called *the optimal hyperplane*. Note that an optimal functional is defined uniquely up to a positive multiplicative constant and the optimal hyperplane is unique [Vapnik, 1998].

the task

We recall, the task of finding an optimal functional is typically formulated as a quadratic programming problem:

$$\begin{array}{ll} \min_{\mathbf{w},b} & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{such that} & y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 \text{ for all } i = 1,\ldots,m \end{array} \tag{6.5}$$

and solved using one of the numerically intensive algorithms developed in the constraint optimization area. The aim of this chapter is to present an alternative solution: a simple local rule, which is a modification of the classical perceptron algorithm, and which we shall call *the maximal margin perceptron (MMP)*.

MMP

6.2.1 Minimal Distance of Convex Hulls

For vectors $\mathbf{w}^{(+1)}, \mathbf{w}^{(-1)} \in \mathbb{R}^N$ it is convenient to introduce the notation

$$\pi_{\mathbf{w}^{(+1)},\mathbf{w}^{(-1)}}(\mathbf{x}) := \pi_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}\cdot\mathbf{x} + b \text{ for all } \mathbf{x}\in\mathbb{R}^N, \tag{6.6}$$

where

$$\mathbf{w} := \mathbf{w}^{(+1)} - \mathbf{w}^{(-1)} \text{ and } b := -\frac{\left\|\mathbf{w}^{(+1)}\right\|^2 - \left\|\mathbf{w}^{(-1)}\right\|^2}{2}. \tag{6.7}$$

support centers

Vectors $\mathbf{w}^{(+1)}$ and $\mathbf{w}^{(-1)}$ as above will be called *support centers* of the hyperplane ${\pi_{\mathbf{w}^{(+1)},\mathbf{w}^{(-1)}}}^{-1}(0)$. This hyperplane is orthogonal to the vector $\mathbf{w}$ and passes through the center $\frac{\mathbf{w}^{(+1)}+\mathbf{w}^{(-1)}}{2}$ of the segment joining $\mathbf{w}^{(+1)}$ to $\mathbf{w}^{(-1)}$ (cf. Figure 6.1).

Let $\mathbf{A}\subset\mathbb{R}^m$ be the collection of all sequences $\vec{\alpha} = (\alpha_i)$ such that $0 \leq \alpha_i \leq 1$, for $i = 1,\ldots,m$ and $\sum_{i\in I^{(+1)}} \alpha_i = \sum_{i\in I^{(-1)}} \alpha_i = 1$. Let

$$\mathbf{w}_{\vec{\alpha}}^{(+1)} := \sum_{i\in I^{(+1)}} \alpha_i \mathbf{x}_i, \tag{6.8}$$

$$\mathbf{w}_{\vec{\alpha}}^{(-1)} := \sum_{i\in I^{(-1)}} \alpha_i \mathbf{x}_i, \tag{6.9}$$

$$\mathbf{w}_{\vec{\alpha}} := \mathbf{w}_{\vec{\alpha}}^{(+1)} - \mathbf{w}_{\vec{\alpha}}^{(-1)} = \sum_{i=1}^{m} y_i\alpha_i\mathbf{x}_i, \tag{6.10}$$

$$\pi_{\vec{\alpha}} := \pi_{\mathbf{w}_{\vec{\alpha}}^{(+1)},\mathbf{w}_{\vec{\alpha}}^{(-1)}} = \pi_{\mathbf{w}_{\vec{\alpha}},b_{\vec{\alpha}}}, \tag{6.11}$$

where

$$b_{\vec{\alpha}} := -\frac{\left\|\mathbf{w}_{\vec{\alpha}}^{(+1)}\right\|^2 - \left\|\mathbf{w}_{\vec{\alpha}}^{(-1)}\right\|^2}{2} = \frac{1}{2}\left(\sum_{i,j\in I^{(-1)}} \alpha_i\alpha_j\mathbf{x}_i\cdot\mathbf{x}_j - \sum_{i,j\in I^{(+1)}} \alpha_i\alpha_j\mathbf{x}_i\cdot\mathbf{x}_j\right).$$

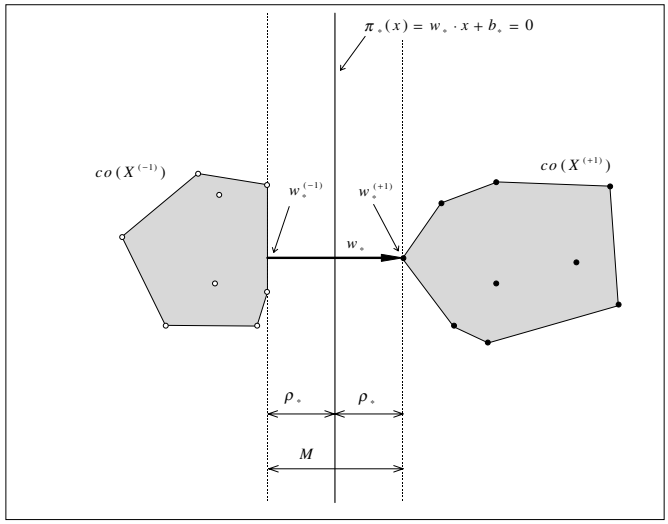


Figure 6.1 Illustration of basic definitions. The optimal hyperplane $\pi_*^{-1}(0)$ is determined by a pair $(\mathbf{w}_*^{(-1)}, \mathbf{w}_*^{(+1)}) \in \mathrm{co}\, X^{(-1)} \times \mathrm{co}\, X^{(+1)}$ of vectors of closest distance between the convex polytopes $\mathrm{co}\, X^{(-1)}$ and $\mathrm{co}\, X^{(+1)}$ defined as the convex hulls of data points with negative and positive labels, respectively. This hyperplane is perpendicular to the vector $\mathbf{w}_* = \mathbf{w}_*^{(+1)} - \mathbf{w}_*^{(-1)}$ and passes through the middle of the segments $[\mathbf{w}_*^{(-1)}, \mathbf{w}_*^{(+1)}]$. Note that the maximal margin ρ, as defined in this book, is half of the *margin* M defined in [Keerthi et al., 1999]. (For consistency, the margin M is also used as the horizontal axis in figures displaying benchmark results in Section 6.6.2 and the Appendix.)

Let $\mathrm{co}\, X^{(+1)} := \{\mathbf{w}_{\vec{\alpha}}^{(+1)} \ ; \ \vec{\alpha} \in \mathbf{A}\}$ and $\mathrm{co}\, X^{(-1)} := \{\mathbf{w}_{\vec{\alpha}}^{(-1)} \ ; \ \vec{\alpha} \in \mathbf{A}\}$ denote the convex hulls of $X^{(+1)}$ and $X^{(-1)}$, respectively. The sets $\mathrm{co}\, X^{(+1)}$ and $\mathrm{co}\, X^{(-1)}$ are compact, since both $X^{(+1)}$ and $X^{(-1)}$ are compact (finite). Hence there exist $\vec{\alpha}_* \in \mathbf{A}$ (not necessarily unique) such that the pair of points

$$(\mathbf{w}_*^{(+1)}, \mathbf{w}_*^{(-1)}) := (\mathbf{w}_{\vec{\alpha}_*}^{(+1)}, \mathbf{w}_{\vec{\alpha}_*}^{(-1)}) \in \mathrm{co}\, X^{(+1)} \times \mathrm{co}\, X^{(-1)}$$

minimizes distance between $\mathrm{co}\, X^{(+1)}$ and $\mathrm{co}\, X^{(-1)}$, i.e.,

$$\|\mathbf{w}_*\| = \left\|\mathbf{w}_*^{(+1)} - \mathbf{w}_*^{(-1)}\right\| = \min_{(\mathbf{x},\mathbf{x}') \in \mathrm{co}\, X^{(+1)} \times \mathrm{co}\, X^{(-1)}} \|\mathbf{x} - \mathbf{x}'\|, \tag{6.12}$$

where

$$\mathbf{w}_* := \mathbf{w}_{\vec{\alpha}_*} = \mathbf{w}_*^{(+1)} - \mathbf{w}_*^{(-1)} = \sum_{i=1}^{m} \alpha_i^* y_i \mathbf{x}_i.$$

basic estimate of margin

Proposition 6.1

(i) If $\mathbf{w}^{(+1)} \in \operatorname{co} X^{(+1)}$ and $\mathbf{w}^{(-1)} \in \operatorname{co} X^{(-1)}$ and $\mathbf{w} = \mathbf{w}^{(+1)} - \mathbf{w}^{(-1)}$, then

$$\rho(\pi_{\mathbf{w}^{(+1)},\mathbf{w}^{(-1)}}) \le \rho \le \left\|\mathbf{w}^{(+1)} - \mathbf{w}^{(-1)}\right\|/2 = \|\mathbf{w}\|/2. \tag{6.13}$$

(ii) $\rho = 0.5 \left\|\mathbf{w}_*^{(+1)} - \mathbf{w}_*^{(-1)}\right\|$.

(iii) The functional $\pi_{\mathbf{w}_*^{(+1)},\mathbf{w}_*^{(-1)}} : \mathbb{R}^N \to \mathbb{R}$ is optimal.

(iv) The vector $\mathbf{w}_* := \mathbf{w}_*^{(+1)} - \mathbf{w}_*^{(-1)}$ is unique.

Proposition 6.1 (i) provides a practical criterion for checking the quality of generated estimates of the optimal functional.

Note that, by Proposition 6.1 (ii) and (iv), every inequality in (6.13) is sharp unless $\mathbf{w}^{(+1)} - \mathbf{w}^{(-1)} = \mathbf{w}_*$, in which case all inequalities become equalities.

Proof

(i) Obviously the first inequality in (6.13) holds by the definition of the maximal margin ρ. We shall show the second one now.
For an optimal functional $\pi'(\mathbf{x}) = \mathbf{x} \cdot \mathbf{w}' + b'$ let us consider the two convex regions (half spaces)

$$S^+ := \{\mathbf{x}\ ;\ \pi'(\mathbf{x}) \ge \|\mathbf{w}'\|\rho\} \text{ and } S^- := \{\mathbf{x}\ ;\ \pi'(\mathbf{x}) \le -\|\mathbf{w}'\|\rho\}.$$

The point $\mathbf{w}^{(+1)}$ belongs to S^+ since

$$\pi'(\mathbf{w}^{(+1)}) = \sum_{i \in I^{(+1)}} \alpha_i \pi'(\mathbf{x}_i) \ge \sum_{i \in I^{(+1)}} \alpha_i \|\mathbf{w}'\|\rho \ge \|\mathbf{w}'\|\rho$$

for an $(\alpha_i) \in \mathbf{A}$ such that $\mathbf{w}^{(+1)} = \sum_{i \in I^{(+1)}} \alpha_i \mathbf{x}_i$; note that $\sum_{i \in I^{(+1)}} \alpha_i = 1$. Likewise, $\mathbf{w}^{(-1)} \in S^-$. Hence

$$\|\mathbf{w}\| = \left\|\mathbf{w}^{(+1)} - \mathbf{w}^{(-1)}\right\| \ge \min_{(\mathbf{x},\mathbf{x}') \in S^+ \times S^-} \|\mathbf{x} - \mathbf{x}'\| = 2\rho,$$

since 2ρ is the distance between hyperplanes S^+ and S^-.

(ii) and (iii) Having shown (6.13), it is sufficient to prove that

$$\rho(\pi_{\mathbf{w}_*^{(+1)},\mathbf{w}_*^{(-1)}}) \ge \left\|\mathbf{w}_*^{(+1)} - \mathbf{w}_*^{(-1)}\right\|/2 = \|\mathbf{w}_*\|/2.$$

Suppose this is not true, hence there exists $i \in \{1, \ldots, m\}$ such that

$$y_i \pi_{\mathbf{w}_*^{(+1)},\mathbf{w}_*^{(-1)}}(\mathbf{x}_i) < \|\mathbf{w}_*\|^2/2.$$

We shall show that this leads to a contradiction. Without loss of generality we can assume $y_i = 1$. Simple algebra shows that the above inequality is equivalent to

$$\mathbf{w}_* \cdot (\mathbf{x}_i - \mathbf{w}_*^{(+1)}) < 0.$$

Now let us consider the segment $(1 - \tau)\mathbf{w}_*^{(+1)} + \tau \mathbf{x}_i$, $0 \le \tau \le 1$, contained in

$\operatorname{co} X^{(+1)}$ and the differentiable function $\phi(\tau) := \left\|(1-\tau)\mathbf{w}_*^{(+1)} + \tau\mathbf{x}_i - \mathbf{w}_*^{(-1)}\right\|^2$ giving the squared distance of the point of the segment from $\mathbf{w}_*^{(-1)} \in \operatorname{co} X^{(-1)}$. Hence, $\phi(\tau) > \|\mathbf{w}_*\|^2$ for $0 \leq \tau \leq 1$, since $\|\mathbf{w}_*\|$ is the distance between $\operatorname{co} X^{(+1)}$ and $\operatorname{co} X^{(-1)}$. However, this contradicts that $\frac{d\phi}{d\tau}(0) = 2\mathbf{w}_* \cdot (\mathbf{x}_i - \mathbf{w}_*^{(+1)}) < 0$, which implies that $\phi(\tau) < \phi(0) = \|\mathbf{w}_*\|^2$ for a sufficiently small $\tau > 0$.

(iv) Indeed, suppose $\mathbf{w}_A^{(+1)} \in \operatorname{co} X^{(+1)}$ and $\mathbf{w}_A^{(-1)} \in \operatorname{co} X^{(-1)}$, are such that $\mathbf{w}_A := \mathbf{w}_A^{(+1)} - \mathbf{w}_A^{(-1)} \neq \mathbf{w}_*$ and $\|\mathbf{w}_A\| = \|\mathbf{w}_*\|$. We demonstrate that a contradiction would follow. Let $\tilde{\mathbf{w}}^{(+1)} := 0.5(\mathbf{w}_*^{(+1)} + \mathbf{w}_A^{(+1)}) \in \operatorname{co} X^{(+1)}$, $\tilde{\mathbf{w}}^{(-1)} := 0.5(\mathbf{w}_*^{(-1)} + \mathbf{w}_A^{(-1)}) \in \operatorname{co} X^{(-1)}$ and $\tilde{\mathbf{w}} := 0.5(\tilde{\mathbf{w}}^{(+1)} - \tilde{\mathbf{w}}^{(-1)}) = 0.5(\mathbf{w}_* + \mathbf{w}_A)$. The last equality implies that $\|\tilde{\mathbf{w}}\| < \|\mathbf{w}_*\| = \|\mathbf{w}_A\|$ since $\|\tilde{\mathbf{w}}\|$ equals the height of the triangle with two sides equal $\|\mathbf{w}_*\|$. This would contradict (6.12). ■

Remark 6.2

It can be shown that the following conditions are equivalent:

(i) Data is linearly separable.

(ii) $\operatorname{co} X^{(+1)} \cap \operatorname{co} X^{(-1)} = \varnothing$.

(iii) $\mathbf{w}_*^{(+1)} \neq \mathbf{w}_*^{(-1)}$.

(iv) $\rho = 0.5 \left\|\mathbf{w}_*^{(+1)} - \mathbf{w}_*^{(-1)}\right\| > 0$.

6.2.2 Approximation Steps

The basic algorithm will approximate the optimal functional $\pi_{\mathbf{w}_*^{(+1)}, \mathbf{w}_*^{(-1)}}$ by a sequence of functionals

$$\pi_t(\mathbf{x}) := \pi_{\mathbf{w}_t^{(+1)}, \mathbf{w}_t^{(-1)}}(\mathbf{x}) = \mathbf{w}_t \cdot \mathbf{x} - \frac{\left\|\mathbf{w}_t^{(+1)}\right\|^2 - \left\|\mathbf{w}_t^{(-1)}\right\|^2}{2}, \tag{6.14}$$

$$\mathbf{w}_t^{(y)} := \mathbf{w}_{\vec{\alpha}_t}^{(y)} \text{ (for } y = \pm 1),$$

where $\vec{\alpha}_t = (\alpha_{t,i}) \in \mathbf{A}$, for $t = 1, 2, \ldots$. The sequence $(\vec{\alpha}_t)$ will be constructed iteratively in such a way that

$$dw^2 := \|\mathbf{w}_t\|^2 - \|\mathbf{w}_{t+1}\|^2 > 0 \text{ for all } t. \tag{6.15}$$

Assume that $\vec{\alpha}_1, \ldots, \vec{\alpha}_t$ have been defined. Then $\vec{\alpha}_{t+1}$ is defined using one of the two steps described below (some geometrical justification is given in Figure 6.2). Let

$$G(\vec{\alpha}_t; i) := y_i(\pi_t(\mathbf{w}_t^{(y_i)}) - \pi_t(\mathbf{x}_i)) = y_i(\mathbf{w}_t^{(y_i)} - \mathbf{x}_i) \cdot \mathbf{w}_t, \tag{6.16}$$

$$H(\vec{\alpha}_t; i) := \left\|\mathbf{w}_t^{(y_i)} - \mathbf{x}_i\right\|, \tag{6.17}$$

for $\vec{\alpha}_t \in \mathbf{A}$, $i \in \{1, \ldots, m\}$ (The function $\mathbf{x}_i \mapsto G(\vec{\alpha}_t; i)$ indicates how the ith instance is scored by π_t with respect to the score for $\mathbf{w}_t^{(y_i)}$. It is obviously defined for any $\vec{\alpha} \in \mathbf{A}$). Note that $G(\vec{\alpha}_t; i)/H(\vec{\alpha}_t; i)$ is the value of the projection of the

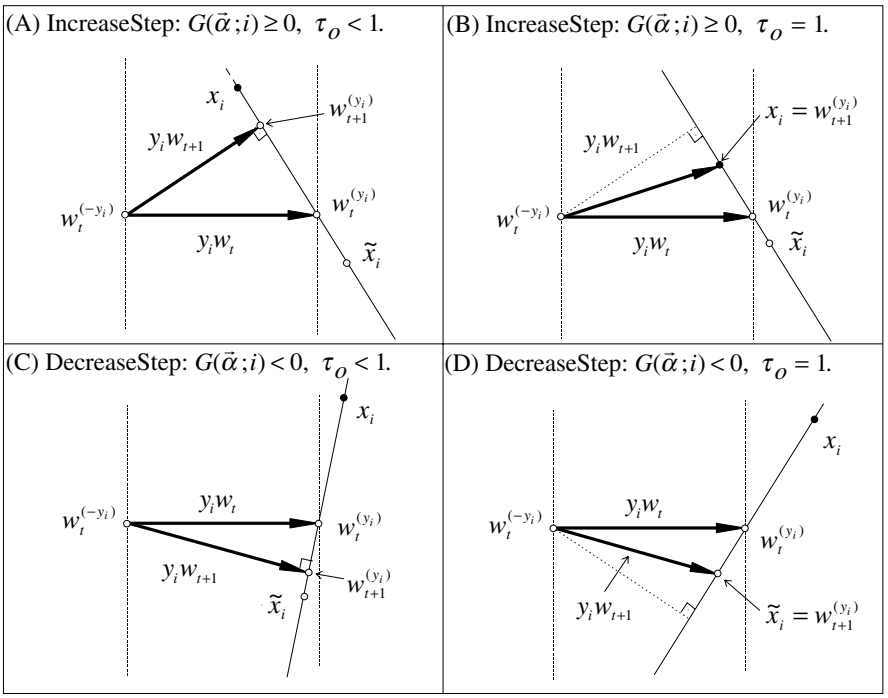


Figure 6.2 Geometrical illustration of the four basic cases of the update steps. The principle is as follows. Let $\tilde{\mathbf{x}}_i := \sum_{i \neq j \in I^{(y_i)}} \alpha_j \mathbf{x}_j / (1 - \alpha_i) \in \operatorname{co} X^{(y_i)}$ denote the point of the convex hull $\operatorname{co} X^{(y_i)}$ obtained by "removing" from $\mathbf{w}_t^{(y_i)} = \sum_{i \in I^{(y_i)}} \alpha_i \mathbf{x}_i$ the contribution from the ith data vector, (and then rescaling accordingly). (Note that $\tilde{\mathbf{x}}_i = \mathbf{w}_t^{(y_i)}$ if $\alpha_i = 0$.) The support center $\mathbf{w}_t^{(y_i)}$ is always shifted to the point $\mathbf{w}_{t+1}^{(y_i)}$ of the segment $[\mathbf{x}_i, \tilde{\mathbf{x}}_i]$ being the closest to the other support center, $\mathbf{w}_t^{(-y_i)}$. This will be the point of the orthogonal projection of $\mathbf{w}_t^{(-y_i)}$ onto the direction $\tilde{\mathbf{x}}_i - \mathbf{x}_i$ (being the same as the direction of $\mathbf{w}_t^{(y_i)} - \mathbf{x}_i$), if the projection falls within the segment $[\mathbf{x}_i, \tilde{\mathbf{x}}_i]$, cf. Figures (A) and (C). However, if the orthogonal projection falls outside of this segment, then $\mathbf{w}_{t+1}^{(y_i)}$ becomes $\mathbf{x}_i$ or $\tilde{\mathbf{x}}_i$, depending which is closest to $\mathbf{w}_t^{(-y_i)}$; cf. Figures (B) and (D).

vector $y_i \mathbf{w}_t$ onto the direction of $\mathbf{w}_t^{(y_i)} - \mathbf{x}_i$. If the projection of the point $\mathbf{w}_t^{(-y_i)}$ onto that direction falls into the segment $[\mathbf{w}_t^{(y_i)}, \mathbf{x}_i]$, then it splits this segment into proportion $\frac{G(\vec{\alpha}_t;i)}{H(\vec{\alpha}_t;i)^2} \ : \ (1 - \frac{G(\vec{\alpha}_t;i)}{H(\vec{\alpha}_t;i)^2})$ (cf. Figure 6.2).

IncreaseStep We chose a training instance $(\mathbf{x}_i, y_i)$ such that $G(\vec{\alpha}_t; i) \geq 0$ and $\mathbf{x}_i \neq \mathbf{w}_t^{(y_i)}$, and then set $\mathbf{w}_{t+1}^{(y_i)}$ to be the point of the segment $\tau \mathbf{x}_i + (1 - \tau)\mathbf{w}_t^{(y_i)}$, $0 \leq \tau \leq 1$, closest to $\mathbf{w}_{t+1}^{(-y_i)} := \mathbf{w}_t^{(-y_i)}$. This is equivalent to taking the point of the

segment for $\tau = \tau_o$, where

$$\tau_o := \min\left(1, \frac{y_i(\mathbf{w}_t^{(y_i)} - \mathbf{x}_i)\cdot \mathbf{w}_t}{\left\|\mathbf{x}_i - \mathbf{w}_t^{(y_i)}\right\|^2}\right) = \min\left(1, \frac{G(\vec{\alpha}_t; i)}{H(\vec{\alpha}_t; i)^2}\right) > 0, \tag{6.18}$$

or to setting

$$\alpha_{t+1,j} := \begin{cases} \tau_o\delta_{ij} + (1-\tau_o)\alpha_{t,j} & \text{if } j \in I^{(y_i)}, \\ \alpha_{t,j} & \text{otherwise.} \end{cases} \tag{6.19}$$

DecreaseStep We choose $i \in \{1, \ldots, m\}$ such that $G(\vec{\alpha}_t; i) < 0$, $\mathbf{x}_i \neq \mathbf{w}_t^{y_i}$ and $0 < \alpha_{t,i} < 1$. Let

$$\beta_{t,i} := \frac{\alpha_{t,i}}{1 - \alpha_{t,i}}. \tag{6.20}$$

We set $\mathbf{w}_{t+1}^{(y_i)}$ to be the point of the segment

$$\tau\frac{\mathbf{w}_t^{(y_i)} - \alpha_{t,i}\mathbf{x}_i}{1-\alpha_{t,i}} + (1-\tau)\mathbf{w}_t^{(y_i)} = \mathbf{w}_t^{(y_i)}(1 + \tau\beta_{t,i}) - \mathbf{x}_i\tau\beta_{t,i},$$

$0 \leq \tau \leq 1$, closest to $\mathbf{w}_{t+1}^{(-y_i)} := \mathbf{w}_t^{(-y_i)}$. This is equivalent to taking the point for $\tau = \tau_o$, where

$$\tau_o := \min\left(1, \frac{y_i(\mathbf{w}^{(y_i)} - \mathbf{x}_i)\cdot \mathbf{w}_t}{\beta_{t,i}\left\|\mathbf{x}_i - \mathbf{w}_t^{(y_i)}\right\|^2}\right) = \min\left(1, \frac{G(\vec{\alpha}_t; i)}{\beta_{t,i}H(\vec{\alpha}_t; i)^2}\right) > 0, \tag{6.21}$$

or to setting

$$\alpha_{t+1,j} := \begin{cases} -\tau_o\beta_{t,i}\delta_{ij} + (1+\tau_o\beta_{t,i})\alpha_{t,j} & \text{if } j \in I^{(y_i)}, \\ \alpha_{t,j} & \text{otherwise.} \end{cases} \tag{6.22}$$

Remark 6.3

(i) Note that the IncreaseStep is increasing the Lagrange multiplier of the ith instance, i.e., $\alpha_{t+1,i} > \alpha_{t,i}$. In the extreme case of $\tau_o = 1$, we get $\alpha_{t+1,i} = 1$ and $\alpha_{t+1,j} = 0$ for $j \in I^{(y_i)}$, $i \neq j$, hence all support vectors $\mathbf{x}_j$, $i \neq j \in I^{(y_i)}$ are "pruned." On the other hand, the DecreaseStep is decreasing the Lagrange multiplier of the ith instance, i.e., $\alpha_{t+1,i} < \alpha_{t,i}$. In the extreme case of $\tau_o = 1$ we have $\alpha_{t+1,i} = 0$ and the support vector $\mathbf{x}_i$ is "pruned."

(ii) The DecreaseStep is equivalent to the IncreaseStep with $\mathbf{x}_i$ replaced by the "virtual" point $\tilde{\mathbf{x}}_i := \frac{\mathbf{w}_t^{(y_i)} - \alpha_{t,i}\mathbf{x}_i}{1-\alpha_{t,i}}$ of the convex hull $\operatorname{co} X^{(y_i)}$ (corresponding to $(\tilde{\alpha}_j) \in \mathbf{A}$ obtained from $\vec{\alpha}_t = (\alpha_{t,i})$ by setting ith coordinate to 0, and then rescaling to satisfy $\sum_{i \in I^{(y_i)}} \tilde{\alpha}_j = 1$).

At any time t, the ith training instant can satisfy a precondition of one and only one of the above two steps.

A straightforward calculation based on simple geometry (cf. Figure 6.2) leads to the following expressions for the decrease (6.15) under the application of one of the above two steps:

$$dw^2(\vec{\alpha}_t; i) := \begin{cases} g^2/h^2 & \text{if } h^2 \geq g > 0 \quad (\equiv \text{IncreaseStep}, \tau_o < 1), \\ 2g - h^2 & \text{if } g > h^2 \quad (\equiv \text{IncreaseStep}, \tau_o = 1), \\ g^2/h^2 & \text{if } \beta h^2 \geq -g > 0 \quad (\equiv \text{DecreaseStep}, \tau_o < 1), \\ 2\beta g - \beta^2 h^2 & \text{if } -g > \beta h^2 \quad (\equiv \text{DecreaseStep}, \tau_o = 1), \\ 0 & \text{otherwise}, \end{cases} \tag{6.23}$$

where $h := H(\vec{\alpha}_t; i)$, $g := G(\vec{\alpha}_t; i)$ and $\beta := \beta_{t,i}$ for $i = 1, \ldots, m$.

Obviously not every decrease $dw^2(\vec{\alpha}_t; i)$ will be significant. It is desired to introduce a trigger condition guaranteeing such a significant decrease and such that if it fails for all training instances, then the solution derived at the time must have a good separation margin. A number of options exists to this end. Our preferred conditions are introduced in the following Lemma.

Lemma 6.4

$\|\mathbf{w}_t\|$ decrease

The inequality

$$\|\mathbf{w}_{t+1}\|^2 \leq \|\mathbf{w}_t\|^2 - \theta^2 \tag{6.24}$$

IncreaseStep trigger

holds if either

$$\theta^2 = \theta_{\text{Incr}}(\vec{\alpha}_t; i) := \min\left(\frac{G(\vec{\alpha}_t; i)^2}{H(\vec{\alpha}_t; i)^2},\ G(\vec{\alpha}_t; i)\right) > 0, \tag{6.25}$$

DecreaseStep trigger

and the IncreaseStep was applied or

$$\theta^2 = \theta_{\text{Decr}}(\vec{\alpha}_t; i) := \chi(\alpha_{t,i}) \min\left(\frac{G(\vec{\alpha}_t; i)^2}{H(\vec{\alpha}_t; i)^2},\ -\beta_{t,i} G(\vec{\alpha}_t; i)\right) > 0, \tag{6.26}$$

and the DecreaseStep was applied, where $\chi(\alpha) := 1$ if $0 < \alpha < 1$ and is $:= 0$, otherwise.

In (6.26) we assume that $\theta_{\text{Decr}}(\vec{\alpha}_t; i) = 0$ if $\alpha_i = 1$.

Note that for any i only one of $\theta_{\text{Incr}}(\vec{\alpha}_t; i) > 0$ or $\theta_{\text{Decr}}(\vec{\alpha}_t; i) > 0$ can hold at a time, and if it holds, then the IncreaseStep or the DecreaseStep, respectively, can be applied.

Remark 6.5

The bound (6.24) in the case IncreaseStep (6.25) is equivalent to the basic bound A.8 in [Keerthi et al., 1999].

Proof We show the implication (6.25)⇒(6.24). Note that for the IncreaseStep

$$\begin{aligned} \mathbf{w}_{t+1} &= y_i(\mathbf{w}_t^{(y_i)} + (\mathbf{x}_i - \mathbf{w}_t^{(y_i)})\tau_o - \mathbf{w}_t^{(-y_i)}) \\ &= \mathbf{w}_t + y_i(\mathbf{x}_i - \mathbf{w}_t^{(y_i)})\tau_o. \end{aligned}$$

"Squaring" we get

$$\|\mathbf{w}_{t+1}\|^2 = \|\mathbf{w}_t\|^2 + \left\|\mathbf{x}_i - \mathbf{w}_t^{(y_i)}\right\|^2 \tau_o^2 + 2y_i\tau_o\mathbf{w}_t \cdot (\mathbf{x}_i - \mathbf{w}_t^{(y_i)}) \tag{6.27}$$

$$= \|\mathbf{w}_t\|^2 + h^2\tau_o^2 - 2g\tau_o, \tag{6.28}$$

where we have used substitutions $h := \left\|\mathbf{x}_i - \mathbf{w}_t^{(y_i)}\right\| = H(\vec{\alpha}_t, i)$ and $g := -y_i(\mathbf{x}_i - \mathbf{w}_t^{(y_i)}) \cdot \mathbf{w}_t = G(\vec{\alpha}_t, i)$.

If $\tau_o < 1$, then $\tau_o = g/h^2$ (cf. Eqn. 6.18). The substitution for τ_o into (6.28) gives

$$\|\mathbf{w}_{t+1}\|^2 = \|\mathbf{w}_t\|^2 - g^2/h^2 \leq \|\mathbf{w}_t\|^2 - \theta^2,$$

since $g^2/h^2 \geq \theta^2$ by (6.25).

If $\tau_o = 1$, which according to (6.18) is possible only if $g \geq h^2$, then $g^2/h^2 \geq g$, $\theta^2 = g$ and (6.28) takes the form

$$\|\mathbf{w}_{t+1}\|^2 = \|\mathbf{w}_t\|^2 + h^2 - 2g \leq \|\mathbf{w}_t\|^2 - g = \|\mathbf{w}_t\|^2 - \theta^2.$$

It remains to prove the implication (6.26)⇒(6.24) for the DecreaseStep. This is quite analogous to the previous part of the proof (details are omitted). Note that alternatively, this part of the proof can be derived from the previous one under the formal substitution $g \leftarrow -\beta_{t,i}g$ and $h \leftarrow \beta_{t,i}h$, which reduces the DecreaseStep to the IncreaseStep with the virtual point $\tilde{\mathbf{x}}_i$ replacing $\mathbf{x}_i$, cf. Remark 6.3.(*ii*) and Figure 6.2. ■

6.2.3 Bounds on Margin

In this subsection we link functions θ_{Incr} and θ_{Decr} with bounds which will be used to quantify the minimal margin achieved on exit from algorithms given in this chapter. Let $\vec{\alpha} \in \mathbf{A}$ and $i \in \{1, \ldots, m\}$. Let us first observe that

$$\left|\frac{G(\vec{\alpha}; i)}{H(\vec{\alpha}; i)}\right| = \left|\frac{\mathbf{w}_{\vec{\alpha}} \cdot (\mathbf{x}_i - \mathbf{w}_{\vec{\alpha}}^{(y_i)})}{\|\mathbf{w}_{\vec{\alpha}}\|\left\|\mathbf{x}_i - \mathbf{w}_{\vec{\alpha}}^{(y_i)}\right\|}\right| \|\mathbf{w}_{\vec{\alpha}}\| \leq \|\mathbf{w}_{\vec{\alpha}}\|, \tag{6.29}$$

$$y_i\pi_{\vec{\alpha}}(\mathbf{x}_i) = \frac{\|\mathbf{w}_{\vec{\alpha}}\|^2}{2} - G(\vec{\alpha}; i), \tag{6.30}$$

$$y_i\pi_{\vec{\alpha}}(\mathbf{w}_{\vec{\alpha}}^{(y_i)}) = \frac{\|\mathbf{w}_{\vec{\alpha}}\|^2}{2}. \tag{6.31}$$

The last two relations are derived by direct algebra. It is convenient to introduce the following definition

$$\delta(\vec{\alpha}, i) := \frac{\alpha_i y_i(\pi_{\vec{\alpha}}(\mathbf{x}_i) - \pi_{\vec{\alpha}}(\mathbf{w}_{\vec{\alpha}}^{(y_i)}))}{\|\mathbf{w}_{\vec{\alpha}}\|} = -\alpha_i \frac{G(\vec{\alpha}; i)}{\|\mathbf{w}_{\vec{\alpha}}\|}, \tag{6.32}$$

where the last relation follows from (6.30) and (6.31). For an optimal $\vec{\alpha} = \vec{\alpha}_* \in \mathbf{A}$ we have

$$\delta(\vec{\alpha}_*, i) = 0 \text{ for all } i. \tag{6.33}$$

This is a form of the known Karush-Kuhn-Tucker condition (cf. Chapter 1), but we do not pursue this connection here. We rather concentrate on estimating how much this relation is violated for a sub-optimal $\vec{\alpha} \in \mathbf{A}$. A simple bound on $\delta(\vec{\alpha}, i)$ is included in the following Lemma.

Lemma 6.6
For every $\vec{\alpha} \in A$ and every $j \in \{1, \ldots, m\}$ the following relations hold

$$\max_i \theta_{\mathrm{Incr}}(\vec{\alpha}, i) \geq 0, \text{ and } \max_i \theta_{\mathrm{Decr}}(\vec{\alpha}, i) \geq 0, \tag{6.34}$$

$$\rho(\pi_{\vec{\alpha}}) = \frac{\|\mathbf{w}_{\vec{\alpha}}\|}{2} - \frac{\max_i G(\vec{\alpha}, i)}{\|\mathbf{w}_{\vec{\alpha}}\|}, \tag{6.35}$$

$$\rho(\pi_{\vec{\alpha}}) \geq \frac{\|\mathbf{w}_{\vec{\alpha}}\|}{2} - \frac{D}{\|\mathbf{w}_{\vec{\alpha}}\|}\sqrt{\max_i \theta_{\mathrm{Incr}}(\vec{\alpha}, i)}, \tag{6.36}$$

$$\frac{D}{\|\mathbf{w}_{\vec{\alpha}}\|}\sqrt{\max_i \theta_{\mathrm{Decr}}(\vec{\alpha}; i)} \geq \delta(\vec{\alpha}, j) \geq -\frac{D}{\|\mathbf{w}_{\vec{\alpha}}\|}\sqrt{\max_i \alpha_i{}^2 \theta_{\mathrm{Incr}}(\vec{\alpha}; i)}, \tag{6.37}$$

where $D := \max_{1 \leq i,j \leq m} \|\mathbf{x}_i - \mathbf{x}_j\|$ is the diameter of the data set.

Proof

(i) For a proof of (6.34) it is sufficient to demonstrate that there exist $i, i' \in \{1, \ldots, m\}$ such that $\alpha_i G(\vec{\alpha}; i) \geq 0$ and $-\alpha_{i'} G(\vec{\alpha}; i') \geq 0$. Since $\alpha_i \geq 0$ for all i, their existence follows from the following relation:

$$\sum_{i=1}^{m} \alpha_i G(\vec{\alpha}; i) = \sum_{i=1}^{m} \alpha_i y_i (\mathbf{w}_{\vec{\alpha}}^{(y_i)} - \mathbf{x}_i) \cdot \mathbf{w}_{\vec{\alpha}} = 0.$$

(ii) Relation (6.35) follows immediately from (6.30) and the definition of margin (cf. Eqn. 6.3).

(iii) We shall show inequality (6.36). First note that

$$H(\vec{\alpha}, i) = \left\| \mathbf{w}_{\vec{\alpha}}^{(y_i)} - \mathbf{x}_i \right\| = \left\| \sum_{j \in I^{(y_i)}} \alpha_j (\mathbf{x}_j - \mathbf{x}_i) \right\| \leq \sum_{j \in I^{(y_i)}} \alpha_j \|\mathbf{x}_j - \mathbf{x}_i\| \leq D \tag{6.38}$$

since $\alpha_j \geq 0$, $\sum_{j \in I^{(y_i)}} \alpha_j = 1$ and $\|\mathbf{x}_i - \mathbf{x}_j\| \leq D$ for every i and every $j \in I^{(y_i)}$. Similarly,

$$\begin{aligned} \|\mathbf{w}_{\vec{\alpha}}\| &= \left\| \sum_{i \in I^{(+1)}} \alpha_i \mathbf{x}_i - \sum_{i \in I^{(-1)}} \alpha_i \mathbf{x}_i \right\| \leq \left\| \sum_{(i,j) \in I^{(+1)} \times I^{(-1)}} \alpha_i \alpha_j (\mathbf{x}_i - \mathbf{x}_j) \right\| \\ &\leq D \sum_{(i,j) \in I^{(+1)} \times I^{(-1)}} \alpha_i \alpha_j = D. \end{aligned}$$

We show now that

$$G(\vec{\alpha}; i) \leq D\sqrt{\theta_{\mathrm{Incr}}(\vec{\alpha}; i)} \text{ if } G(\vec{\alpha}; i) \geq 0 \tag{6.39}$$

First, if $\theta_{\mathrm{Incr}}(\vec{\alpha};i) = \frac{G(\vec{\alpha};i)^2}{H(\vec{\alpha};i)^2} \leq G(\vec{\alpha};i)$, then

$$G(\vec{\alpha};i) \leq D\frac{G(\vec{\alpha};i)}{H(\vec{\alpha};i)} = D\sqrt{\theta_{\mathrm{Incr}}(\vec{\alpha};i)}$$

by (6.38) and definition (6.25). Next, if $0 \leq \theta_{\mathrm{Incr}}(\vec{\alpha};i) = G(\vec{\alpha};i) \leq \frac{G(\vec{\alpha};i)^2}{H(\vec{\alpha};i)^2}$, then from (6.29) it follows that

$$0 \leq \frac{G(\vec{\alpha};i)}{\|\mathbf{w}_{\vec{\alpha}}\|^2} = \frac{\theta_{\mathrm{Incr}}(\vec{\alpha};i)}{\|\mathbf{w}_{\vec{\alpha}}\|^2} \leq \frac{G(\vec{\alpha};i)^2}{\|\mathbf{w}_{\vec{\alpha}}\|^2 H(\vec{\alpha};i)^2} \leq 1.$$

Hence, $\frac{G(\vec{\alpha};i)}{\|\mathbf{w}_{\vec{\alpha}}\|^2} \leq \frac{\sqrt{\theta_{\mathrm{Incr}}(\vec{\alpha};i)}}{\|\mathbf{w}_{\vec{\alpha}}\|}$, and again

$$G(\vec{\alpha};i) \leq \|\mathbf{w}_{\vec{\alpha}}\|\sqrt{\theta_{\mathrm{Incr}}(\vec{\alpha};i)} \leq D\sqrt{\theta_{\mathrm{Incr}}(\vec{\alpha};i)},$$

since $\|\mathbf{w}_{\vec{\alpha}}\| \leq D$. This completes the proof of (6.39).
Substituting (6.39) into (6.30) we obtain the bound

$$y_i\pi_{\vec{\alpha}}(\mathbf{x}_i) \geq \frac{\|\mathbf{w}_{\vec{\alpha}}\|^2}{2} - D\sqrt{\theta_{\mathrm{Incr}}(\vec{\alpha};i)} \text{ if } \theta_{\mathrm{Incr}}(\vec{\alpha};i) \geq 0.$$

which combined with (6.34) implies (6.36).

(iv) Now we demonstrate bounds (6.37). The lower bound follows from (6.32), (6.39) and (6.34). The proof of the upper bound is quite similar to the previous part of the proof and will be given now. Note that it holds for $\alpha_i = 1$ since $\theta_{\mathrm{Decr}}(\vec{\alpha};i) = 0$ and $\mathbf{x}_i = \mathbf{w}^{(y_i)}$, thus $\delta(\vec{\alpha},i) = 0$, in such a case. Hence assume $0 \leq \alpha_i < 1$ for the rest of the proof. If $0 < \theta_{\mathrm{Decr}}(\vec{\alpha};i) = G(\vec{\alpha};i)^2/H(\vec{\alpha};i)^2 \leq -\beta_i G(\vec{\alpha};i)$, then

$$0 \leq -\alpha_i G(\vec{\alpha};i) \leq -G(\vec{\alpha};i) \leq -DG(\vec{\alpha};i)/H(\vec{\alpha};i) = D\sqrt{\theta_{\mathrm{Decr}}(\vec{\alpha};i)},$$

because $H \leq D$ by (6.38).
If $0 \leq \theta_{\mathrm{Decr}}(\vec{\alpha};i) = -\beta_i G(\vec{\alpha};i) < G(\vec{\alpha};i)^2/H(\vec{\alpha};i)^2$, then from(6.29):

$$0 \leq -\frac{\alpha_i G(\vec{\alpha};i)}{\|\mathbf{w}_{\vec{\alpha}}\|^2} \leq -\frac{\beta_i G(\vec{\alpha};i)}{\|\mathbf{w}_{\vec{\alpha}}\|^2} = \frac{\theta_{\mathrm{Decr}}(\vec{\alpha};i)}{\|\mathbf{w}_{\vec{\alpha}}\|^2} \leq 1.$$

Hence,

$$-\frac{\alpha_i G(\vec{\alpha};i)}{\|\mathbf{w}_{\vec{\alpha}}\|^2} \leq \frac{\sqrt{\theta_{\mathrm{Decr}}(\vec{\alpha};i)}}{\|\mathbf{w}_{\vec{\alpha}}\|}$$

and again

$$-\alpha_i G(\vec{\alpha};i) \leq \|\mathbf{w}_{\vec{\alpha}}\|\sqrt{\theta_{\mathrm{Decr}}(\vec{\alpha};i)} \leq D\sqrt{\theta_{\mathrm{Decr}}(\vec{\alpha};i)}.$$

Putting the above bounds on $-\alpha_i G(\vec{\alpha}; i)$ together with the definition (6.32) of δ, we get

$$\delta(\vec{\alpha}, i) \leq \frac{D}{\|\mathbf{w}_{\vec{\alpha}}\|}\sqrt{\theta_{\mathrm{Decr}}(\vec{\alpha}; i)},$$

whenever $G(\vec{\alpha}; i) < 0$. This in combination with (6.34) gives the upper bound in (6.37). ■

6.3 Basic Algorithms

In this section we formulate a number of algorithms for generation of support vector machines and present proofs of convergence. We start with the basic maximal margin perceptron algorithm which can be used for both on-line and batch learning.

Algorithm 6.1 Basic MMP

Given: A method of searching for the next index i_{t+1} (cf. Remark 6.7).
1. *Choose* θ_o, $0 < \theta_o < 1$, $\vec{\alpha}_0 \in \mathbf{A}$. Initialize $t = 0$.
2. *Repeat while either 2.1 or 2.2 can be satisfied*:
   Set $t = t + 1$ and find $i = i_{t+1} \in \{1, \ldots, m\}$ such that either:
   2.1 $\theta_{\mathrm{Incr}}(\vec{\alpha}_t; i) \geq \|\mathbf{w}_t\|^2\theta_o^2$ and define $\vec{\alpha}_{t+1}$ by (6.19) (an IncreaseStep),
   or
   2.2 $\theta_{\mathrm{Decr}}(\vec{\alpha}_t; i) \geq \|\mathbf{w}_t\|^2\theta_o^2$ and define $\vec{\alpha}_{t+1}$ by (6.22) (a DecreaseStep).

Remark 6.7
Several variations of the above algorithm are possible according to the method of selection of the next index i_{t+1} at $t+1$st stage of the Algorithm. Some of them are listed below.

on-line MMP

(i) An "on-line" search with a single pass through the data, with only one instance considered at a time, $i_{t+1} := i_t + 1 \leq m$. In such a case only the first option, IncreaseStep, will be effective. It is also sufficient to keep track only of the upgraded support centers $\mathbf{w}_{t+1}^{(y_i)} := (1 - \tau_o)\mathbf{w}_t^{(y_i)}$ and $\mathbf{w}_{t+1}^{(-y_i)} := \mathbf{w}_t^{(-y_i)}$, where τ_o is given by (6.18) rather than storing Lagrange multipliers $\vec{\alpha}_t$.

(ii) Linear search with multiple passes through the data,
$i_{t+1} := (i_t + 1)$ modulo m.

(iii) Greedy search through the whole data set, with

$$i_{t+1} := \arg\max_j \max\left(\theta_{\mathrm{Incr}}(\vec{\alpha}_t; j), \theta_{\mathrm{Decr}}(\vec{\alpha}_t; j)\right).$$

(iv) Greedy search through the whole data set, with i_{t+1} chosen to maximize (6.23)

$$i_{t+1} := \arg\max_j dw^2(j).$$

(v) Greedy search through the whole data set, with

$$i_{t+1} := \arg\max_j G(\vec{\alpha}_t; j).$$

This is the heuristic used in [Gilbert, 1966]. Note that only IncreseStep is utilized in this case and $\theta_{\rm Incr}$ plays a role only in the stopping criterion.

convergence of Basic MMP

Theorem 6.8

(i) Algorithm 6.1 halts after $t_{\rm upd}$ updates (D is the diameter of the data set) with

$$t_{\rm upd} \leq 2\theta_o^{-2} \ln(\|\mathbf{w}_0\|/2\rho) \leq 2\theta_o^{-2} \ln(D/2\rho). \tag{6.40}$$

(ii) If for $\vec{\alpha}_t$ there does not exist any $i \in \{1, \ldots, m\}$ such that either Steps 2.1 or 2.2 of the algorithm can be satisfied, then $\rho(\pi_{\vec{\alpha}_t}) \geq \rho - D\theta_o$ and $-\alpha_i D\theta_o \leq \delta(\vec{\alpha}_t, i) \leq D\theta_o$, for every $1 \leq i \leq m$.

We recall that ρ is the maximal margin of linear separation of the data. Observe that m does not enter in any of the above bounds, hence the algorithm stops updating $\vec{\alpha}_t$ after a finite number of updates as long as $\rho > 0$, even if $m = \infty$. Note that for $\theta_o := \rho\epsilon/D$ the above theorem implies that after $2D^2\epsilon^{-1}\rho^{-2} \ln(D/2\rho)$ updates we achieve the separation margin $\geq (1-\epsilon)\rho$.

Proof

(i) From (6.24) it follows that for each update in Algorithm 6.1 $\|\mathbf{w}_t\|^2 \leq \|\mathbf{w}_{t-1}\|^2(1-\theta_o^2)$ for $t = 2, 3, \ldots$. Hence after t updates $\|\mathbf{w}_t\|^2 \leq \|\mathbf{w}_0\|^2(1-\theta_o^2)^t$. Since $\rho \leq \|\mathbf{w}_t\|/2$ (cf. Eqn. 6.13), we have $4\rho^2 \leq \|\mathbf{w}_0\|^2(1-\theta_o^2)^t$ and after taking the logarithm of both sides we obtain

$$t \ln(1-\theta_o^2) \geq 2 \ln \frac{2\rho}{\|\mathbf{w}_0\|}.$$

Since $\ln(1-\theta_o^2) \leq -\theta_o^2 < 0$, we get finally

$$t \leq \frac{2 \ln \frac{2\rho}{\|\mathbf{w}_0\|}}{\ln(1-\theta_o^2)} \leq \frac{-2 \ln \frac{2\rho}{\|\mathbf{w}_0\|}}{\theta_o^2}.$$

The first bound in Theorem 6.8 follows. For the second one note that $\|\mathbf{w}_0\| \leq D$.

(ii) If no update can be implemented, then $\max_i \theta_{\rm Incr}(\vec{\alpha}_t, i) < \theta_o^2\|\mathbf{w}_t\|^2$ and $\max_i \theta_{\rm Decr}(\vec{\alpha}_t, i) < \theta_o^2\|\mathbf{w}_t\|^2$. Using these bounds in (6.36) and (6.37) together with the bound $2\rho \leq \|\mathbf{w}_t\|$ (cf. Eqn. 6.13) completes the proof. ■

Now we shall consider a modification of the previous algorithm. The difference with respect to the previous algorithm is that thresholds triggering updates are fixed, independent of $\|\mathbf{w}_t\|$.

Algorithm 6.2 Basic Algorithm

Given: A method of searching for the next index i_{t+1} (cf. Remark 6.7).
1. *Choose* θ_o, $\theta_o > 0$, $\vec{\alpha}_0 \in \mathbf{A}$. Initialize $t = 0$.
2. *Repeat while either 2.1 or 2.2 can be satisfied*:
   Set $t = t + 1$ and find $i = i_{t+1} \in \{1, \ldots, m\}$ such that either:
   2.1 $\theta_{\text{Incr}}(\vec{\alpha}_t; i) \geq \theta_o^2$ and define $\vec{\alpha}_{t+1}$ by (6.19) (an IncreaseStep),
   or
   2.2 $\theta_{\text{Decr}}(\vec{\alpha}_t; i) \geq \theta_o^2$ and define $\vec{\alpha}_{t+1}$ by (6.22) (a DecreaseStep).

convergence of Algorithm 6.2

Theorem 6.9

(i) The Algorithm 6.2 halts after

$$t_{\text{upd}} \leq (\|\mathbf{w}_0\|^2 - 4\rho^2)\theta_o^{-2} \leq D^2\theta_o^{-2} \tag{6.41}$$

updates, where D is the diameter of the data set.

(ii) If after t updates no $i \in \{1, \ldots, m\}$ can satisfy condition 2.1 or 2.2 of Algorithm 6.2, then

$$\rho(\pi_{\vec{\alpha}_t}) \geq \rho - D\theta_o/(2\rho) \tag{6.42}$$

and

$$-\alpha_i D\theta_o/(2\rho) \leq \delta(\vec{\alpha}_t, i) \leq D\theta_o/(2\rho) \text{ for every } 1 \leq i \leq m. \tag{6.43}$$

Proof

(i) After t updates by Algorithm 6.2, $\|\mathbf{w}_t\|^2 \leq \|\mathbf{w}_0\|^2 - t\theta_o^2$ (by Eqn. 6.24), hence

$$t \leq (\|\mathbf{w}_0\|^2 - \|\mathbf{w}_t\|^2)\theta_o^{-2} \leq (\|\mathbf{w}_0\|^2 - 4\rho^2)\theta_o^{-2}.$$

(ii) If no update in Algorithm 6.2 is possible, then

$$\max_i \theta_{\text{Incr}}(\vec{\alpha}_t, i) < \theta_o^2 \text{ and } \max_i \theta_{\text{Decr}}(\vec{\alpha}_t, i) < \theta_o^2. \tag{6.44}$$

Hence from (6.36) we have

$$\rho(\pi_{\vec{\alpha}_t}) \geq \frac{\|\mathbf{w}_t\|}{2} - \frac{D}{\|\mathbf{w}_t\|}\theta_o \geq \rho - \frac{D}{2\rho}\theta_o \tag{6.45}$$

since $\rho \leq \|\mathbf{w}_t\|/2$ according to (6.13). Bounds on $\delta(\vec{\alpha}, i)$ follow from (6.37), (6.44) and (6.13). ■

Remark 6.10

The bound (6.42) in Theorem 6.9 is achievable in not more than $2\frac{\rho^2}{\theta_o^2} \ln \frac{\|\mathbf{w}_0\|}{2\rho}$ updates by Algorithm 6.1 according to Theorem 6.8, which is much smaller than the bound $\leq \frac{\|\mathbf{w}_0\|^2 - \rho^2}{\theta_o^2}$ on the number updates of required by Algorithm 6.2 provided by Theorem 6.9, if $\|\mathbf{w}_0\| \gg \rho$. This is the reason why we have followed the line of Algorithm 6.1 in the rest of this chapter.

6.3.1 Exit with Guaranteed Separation Margin

Exit from Algorithm 6.1 is determined by the choice of θ_o (if multiple passes through a finite data set are allowed). If θ_o is sufficiently small then this yields a functional with an appreciable margin $\geq \rho - D\theta_o$. However, up front choice of θ_o may not be obvious, especially if ρ and D are not known. In this section we describe a way around this obstacle in the case of batch learning. It is based on a practical stopping criterion guaranteeing an approximation of the maximal margin with a predefined precision. In order to formalize this we define *the precision of margin approximation (POMA)* for $\vec{\alpha} \in \mathbf{A}$:

precision of margin approximation (POMA)

$$\epsilon(\vec{\alpha}) := \frac{\rho - \rho(\pi_{\vec{\alpha}})}{\rho}. \tag{6.46}$$

Lemma 6.11
Let $0 < \epsilon < 1$ and $\vec{\alpha} \in \mathbf{A}$. If $\max_{i=1,\ldots,m} \frac{2G(\vec{\alpha};i)}{\|\mathbf{w}\|^2} \leq \epsilon$, then $\rho(\pi_{\vec{\alpha}}) \geq \rho(1-\epsilon)$, hence $\epsilon(\vec{\alpha}) \leq \epsilon$.

Proof This is implied directly by the following expression on the margin

$$\rho(\pi_{\vec{\alpha}}) = \frac{\|\mathbf{w}_{\vec{\alpha}}\|}{2} - \max_i \frac{G(\vec{\alpha};i)}{\|\mathbf{w}_{\vec{\alpha}}\|}$$

(cf. Eqn. 6.35) and the bound $\rho \leq \|\mathbf{w}_{\vec{\alpha}}\|/2$ (cf. Eqn. 6.13). ■

We recall, in the following algorithm the symbol $dw^2(\vec{\alpha}_t; j)$ denotes the decrease $\|\mathbf{w}_t\|^2 - \|\mathbf{w}_{t+1}\|^2$ under assumption that the update $\mathbf{w}_{t+1}$ was obtained by applying the IncreaseStep or the DecreaseStep, respectively, to jth data point (cf. Eqn. 6.23).

Algorithm 6.3 Greedy MMP

1. *Choose* target POMA ϵ, $0 < \epsilon < 1$, $\vec{\alpha}_0 \in \mathbf{A}$ and $t = 1$.
2. *Repeat while* $\max_{i=1,\ldots,m} \frac{2G(\vec{\alpha}_t,i)}{\|\mathbf{w}_t\|^2} > \epsilon$:
   Define α_{t+1} using, respectively, the IncreaseStep (6.19) or the DecreaseStep (6.22) for $i := \arg\max_j dw^2(\vec{\alpha}_t; j)$.
   Reset $t \leftarrow t + 1$.

convergence of Greedy MMP

Theorem 6.12
Algorithm 6.3, halts after

$$t_{\text{upd}} \leq \frac{2D^2}{\rho^2\epsilon^2} \ln \frac{\|\mathbf{w}_o\|}{2\rho} \leq \frac{2D^2}{\rho^2\epsilon^2} \ln \frac{D}{2\rho} \tag{6.47}$$

updates yielding the separation margin $\geq (1-\epsilon)\rho$.

Proof The bound on the separation margin comes from Lemma 6.11. It remains to show (6.47).

Let $\theta_o := \epsilon\rho/D$ and $\mathbf{w}_t$, $1 \le t \le \tilde{t}$, be the maximal sequence of vectors $\mathbf{w}_t := \mathbf{w}_{\vec{\alpha}_t}$ generated by the Algorithm 6.3 such that

$$\max_i \theta_{\mathrm{Incr}}(\vec{\alpha}_t, i) \ge \theta_o^2 \|\mathbf{w}_t\|^2 \text{ for all } t = 1, ..., \tilde{t} - 1, \tag{6.48}$$

where $\tilde{t} \ge 1$ (possibly $\tilde{t} = \infty$). From Lemma 6.4 it follows that $\|\mathbf{w}_t\|^2 \le \|\mathbf{w}_{t-1}\|^2(1-\theta_o^2)$, hence $\|\mathbf{w}_t\|^2 \le \|\mathbf{w}_0\|^2(1-\theta_o^2)^t$ for $t = 1, 2, 3, \ldots, \tilde{t}$. Since $\|\mathbf{w}_t\| \ge 2\rho$, we find as in the proof of Theorem 6.8(i), that

$$\tilde{t} \le \frac{2D^2}{\rho^2\epsilon^2} \ln \frac{D}{2\rho}. \tag{6.49}$$

Hence $\tilde{t}$ is finite and the bound (6.48) does not hold for $t = \tilde{t}$, i.e.,

$$\max_i \theta_{\mathrm{Incr}}(\vec{\alpha}_{\tilde{t}}, i) < \theta_o^2 \|\mathbf{w}_{\tilde{t}}\|^2.$$

Now from Eqn. 6.36 of Lemma 6.6 we obtain

$$\rho(\pi_{\vec{\alpha}_{\tilde{t}}}) \ge \frac{\|\mathbf{w}_{\tilde{t}}\|}{2} - \frac{D}{\|\mathbf{w}_{\tilde{t}}\|}\theta_o\|\mathbf{w}_{\tilde{t}}\| = \frac{\|\mathbf{w}_{\tilde{t}}\|}{2} - \epsilon\rho \ge \frac{\|\mathbf{w}_{\tilde{t}}\|}{2}(1-\epsilon).$$

After substitution for $\rho(\pi_{\vec{\alpha}_{\tilde{t}}})$ from (6.35) we get

$$\frac{\|\mathbf{w}_{\tilde{t}}\|}{2} - \frac{\max_i G(\vec{\alpha}_{\tilde{t}}; i)}{\|\mathbf{w}_{\tilde{t}}\|} \le \frac{\|\mathbf{w}_{\tilde{t}}\|}{2}(1-\epsilon)$$

which yields $\frac{2\max_i G(\vec{\alpha}_{\tilde{t}};i)}{\|\mathbf{w}_{\tilde{t}}\|^2} \le \epsilon$. Thus $t_{\mathrm{upd}} \le \tilde{t}$ and (6.49) implies (6.47). ■

The above theorem implies immediately the following result.

Corollary 6.13
For the sequence of POMA values for $\vec{\alpha}_t \in \mathbf{A}$ generated by Algorithm 6.3 the following upper bound holds:

$$\epsilon(\vec{\alpha}_t) \le \frac{D}{\rho}\sqrt{\frac{2}{t}\ln\frac{\|\mathbf{w}_o\|}{2\rho}} \le \frac{D}{\rho}\sqrt{\frac{2}{t}\ln\frac{D}{2\rho}} \text{ for all } t = 1, \ldots, t_{\mathrm{upd}}. \tag{6.50}$$

Remark 6.14
It is not hard to observe that the above Theorem and Corollary (and their proofs) hold also for a modication of Algorithm 6.3 in which the selection $i = \arg\max_j dw^2(\vec{\alpha}_t; j)$ in Step 2 is replaced by

$$i = \arg\max_j(\max(\theta_{\mathrm{Incr}}(\vec{\alpha}_t; j), \theta_{\mathrm{Decr}}(\vec{\alpha}_t; j))$$

or even by $i = \arg\max_j \theta_{\mathrm{Incr}}(\vec{\alpha}_t; j)$.

The following algorithm is proposed for those situations where the greedy search for an update in Algorithm 6.3 is not practical. It combines features of the on-line search of Algorithm 6.1 with a guarantee of POMA. It evaluates and selects

candidate support vectors linearly using an acceptance threshold θ which is lowered only gradually. This is done in order to control proliferation of a number of "low quality" support vectors.

Algorithm 6.4 Linear MMP

Given ϵ, $0 < \epsilon < 1$, $F > 1$, $\vec{\alpha} \in \mathbf{A}$, $\theta_1 > 0$.
1. *Initialize* $G_{\max} = \gamma_{\max} = 0$, $\theta = \theta_1$, $i = 0$ and $G_{\mathbf{w}} = 1$.
2. *While* $G_{\mathbf{w}} > \epsilon$ repeat the following two steps:
   2.1. For $i \leftarrow (i+1)$ modulo m:
      (i) Let $G_{\max} = \max(G_{\max}, G(\vec{\alpha}; i))$, $\gamma = \max\left(\theta_{\mathrm{Incr}}(\vec{\alpha}; i), \theta_{\mathrm{Decr}}(\vec{\alpha}; i)\right)$ and if $\gamma > \gamma_{\max}$, then $\gamma_{\max} = \gamma$ and $i_{\max} = i$.
      (ii) If $\gamma > \theta^2 \|\mathbf{w}_{\vec{\alpha}}\|^2$, then update $\vec{\alpha}$ using the IncreaseStep (6.19) or the DecreaseStep (6.22), respectively, and reset $G_{\max} = 0$ and $\gamma_{\max} = 0$.
   2.2. If no update was made for last m checks of 2.1(ii), then reset $\theta = \sqrt{\gamma_{\max}}/(F\|\mathbf{w}_{\vec{\alpha}}\|)$, $G_{\mathbf{w}} = 2G_{\max}/\|\mathbf{w}_{\vec{\alpha}}\|^2$ and $i = i_{\max} - 1$.

Note that for each successive set of $G_{\max}$ and $\gamma_{\max}$ defined in the Step 2.2 of the above algorithm we have $G_{\max} = \max_j G(\vec{\alpha}; j)$ and

$$\gamma_{\max} = \max_j \max\left(\theta_{\mathrm{Incr}}(\vec{\alpha}; j), \theta_{\mathrm{Decr}}(\vec{\alpha}; j)\right) \geq 0,$$

where the last inequality follows from (6.34). Additionally, resetting i to $i = i_{\max} - 1$ after change of θ ensures that the algorithm starts with an update.

convergence of Linear MMP

Theorem 6.15
Algorithm 6.4, halts after

$$t_{\mathrm{upd}} \leq 2\left(\min(\frac{\rho\epsilon}{DF},\ \theta_1)\right)^{-2} \ln\frac{D}{2\rho} \tag{6.51}$$

updates yielding the separation margin $\geq (1-\epsilon)\rho$, where θ_1 is the starting values of the threshold θ.

Proof The algorithm stops only if $\max_i G(\vec{\alpha}, i)/\|\mathbf{w}_{\vec{\alpha}}\|^2 = G_{\mathbf{w}} \leq \epsilon$, hence exactly as in the proof of the previous theorem one can demonstrate that on exit $\rho(\pi_{\vec{\alpha}}) \geq (1-\epsilon)\rho$. It remains to show bound (6.51).

For the clarity of the proof it is convenient to introduce the maximal sequence (θ_n), $1 \leq n < \tilde{n}+1$ of successive values of thresholds θ generated by the Algorithm 6.4 as it runs (we shall use subscript $n = 1$ for starting values; we also allow $\tilde{n} = \infty$, at this stage, although we shall demonstrate that always $\tilde{n} < \infty$). It is also convenient to introduce sequences of corresponding parameters: $\gamma_{\max}^{(n)}$, $G_{\mathbf{w}}^{(n)}$, $G_{\max}^{(n)}$, $\mathbf{w}_n^{(+1)}$, $\mathbf{w}_n^{(-1)}$ and $\mathbf{w}_n$ being snapshots of $\gamma_{\max}$, $G_{\mathbf{w}}$, $G_{\max}$, $\mathbf{w}_{\vec{\alpha}}^{(+1)}$, $\mathbf{w}_{\vec{\alpha}}^{(-1)}$ and $\mathbf{w}_{\vec{\alpha}}$, respectively, at the time when the nth value of θ, $\theta = \theta_n$, is introduced in

Algorithm 6.4, for $2 \leq n < \tilde{n}+1$. Similarly we introduce the sequences t_n and $\rho(\pi_n)$ of the number of updates and achieved separation margins $\rho(\pi_{\vec{\alpha}})$ for $1 \leq n < \tilde{n}+1$. Note that $\tilde{n} \geq 2$ and $\tilde{n} = \infty$ if the algorithm does not stop; if $\tilde{n} < \infty$, then $t_{\rm upd} = t_{\tilde{n}}$ since the algorithms terminates only after $\theta_{\tilde{n}}$ was introduced.

From the definition of $\gamma^{(n)}_{\max}$ and θ_n we get

$$\theta_n = \frac{\sqrt{\gamma^{(n)}_{\max}}}{\|\mathbf{w}_n\| F} < \frac{\theta_{n-1}}{F} \tag{6.52}$$

$$\max_i(\theta_{\rm Incr}(\vec{\alpha}_n; i), \theta_{\rm Decr}(\vec{\alpha}_n; i)) \leq \gamma^{(n)}_{\max} \leq \theta^2_{n-1}\|\mathbf{w}_n\|^2 \text{ for all } n > 1. \tag{6.53}$$

Using (6.53), we can show as in the proof of Theorem 6.8(i) that

$$t_n \leq \frac{2\ln(\|\mathbf{w}_1\|/2\rho)}{\theta^2_{n-1}} \text{ for all } n > 2. \tag{6.54}$$

Similarly, from (6.53) and (6.36) it follows that

$$\rho(\pi_n) \geq \frac{\|\mathbf{w}_n\|}{2} - \frac{D\sqrt{\gamma^{(n)}_{\max}}}{\|\mathbf{w}_n\|} \text{ for all } n > 2. \tag{6.55}$$

Now on a substitution for $\rho(\pi_n)$ from (6.35) and using (6.52), we find that

$$\frac{G^{(n)}_{\max}}{\|\mathbf{w}_n\|} \leq \frac{D\sqrt{\gamma^{(n)}_{\max}}}{\|\mathbf{w}_n\|} \leq D\theta_{n-1} \leq \frac{D\theta_1}{F^{n-2}} \text{ for all } n \geq 2. \tag{6.56}$$

From the above inequalities it follows that Algorithm 6.4 must terminate after a finite number of values θ_n was generated, i.e., $\tilde{n} < \infty$. Indeed, suppose $\tilde{n} = \infty$, take any $n > 2 + \ln\frac{D\theta_1}{\epsilon\rho}/\ln F$. Using the bound $2\rho \leq \|\mathbf{w}_n\|$ (cf. Proposition 6.1) we would have

$$\frac{G^{(n)}_{\mathbf{w}}\|\mathbf{w}_n\|}{2} = \frac{G^{(n)}_{\max}}{\|\mathbf{w}_n\|} \leq \frac{D\theta_1}{F^{n-2}} < \epsilon\rho \leq \frac{\epsilon\|\mathbf{w}_n\|}{2},$$

which would give $G^{(n)}_{\mathbf{w}} \leq \epsilon$. Thus the "While" loop in Algorithm 6.4 has to have terminated, at the latest, immediately after the θ_n was introduced, giving $\tilde{n} < \infty$, in spite of our supposition to the contrary.

Now we concentrate on estimation of the total number of updates. If $\tilde{n} = 2$, then directly from (6.54) we have

$$t_{\rm upd} = t_2 \leq \frac{2\ln(\|\mathbf{w}_1\|/2\rho)}{(\theta_1)^2}. \tag{6.57}$$

Hence assume now that $\tilde{n} \geq 3$. Since $\tilde{n}$th value of θ has been generated by the algorithm, it did not terminate after $\theta_{\tilde{n}-1}$ was introduced. This means that the condition of the "While" loop was satisfied at the time $n = \tilde{n} - 1$, i.e.,

$$\epsilon < G^{(\tilde{n}-1)}_{\mathbf{w}} = \frac{2G^{(\tilde{n}-1)}_{\max}}{\|\mathbf{w}_{\tilde{n}-1}\|^2}.$$

On the other hand, from (6.56) we get

$$G_{\max}^{(\tilde{n}-1)} \leq D\sqrt{\gamma_{\max}^{(\tilde{n}-1)}}.$$

Putting this all together we find $\epsilon < 2D\sqrt{\gamma_{\max}^{(\tilde{n}-1)}}/\|\mathbf{w}_{\tilde{n}-1}\|^2$ and

$$\sqrt{\gamma_{\max}^{(\tilde{n}-1)}} \geq \frac{\epsilon||\mathbf{w}_{\tilde{n}-1}||^2}{2D}.$$

Finally,

$$\theta_{\tilde{n}-1} = \frac{\sqrt{\gamma_{\max}^{(\tilde{n}-1)}}}{F\|\mathbf{w}_{\tilde{n}-1}\|} > \frac{\epsilon\|\mathbf{w}_{\tilde{n}-1}\|}{2DF} \geq \frac{\epsilon\rho}{DF},$$

since $\|\mathbf{w}_{\tilde{n}-1}\| \geq 2\rho$, cf. Proposition 6.1. Hence from (6.54) we obtain the bound on the total number of updates (if $\tilde{n} \geq 2$):

$$t_{\text{upd}} = t_{\tilde{n}} \leq \frac{2\ln(\|\mathbf{w}_1\|/2\rho)}{\left(\frac{\epsilon\rho}{DF}\right)^2} = \frac{2D^2F^2\ln(\|\mathbf{w}_1\|/2\rho)}{\epsilon^2\rho^2}.$$

It is easy to see that the above bound and (6.57) can be combined into (6.51). ■

In parallel to Corollary 6.13 the above theorem implies immediately the following bound on the precision of margin approximation (6.46).

Corollary 6.16
If $\theta_1 \geq \frac{\rho\epsilon}{DF}$, then

$$\epsilon(\vec{\alpha}_t) \leq \frac{DF}{\rho}\sqrt{\frac{2}{t}\ln\frac{D}{2\rho}} \text{ for } t = 1, \ldots, t_{\text{upd}}. \tag{6.58}$$

6.4 Kernel Machine Extension

In this section we extend the above "linear" algorithms to the kernel case in the standard way. Assume that $\Phi : \mathbb{R}^N \to Z$ is a mapping from the input space into a vector space Z (*the features space*). Assume that $(\mathbf{z}_1, \mathbf{z}_2) \mapsto \mathbf{z}_1 \cdot \mathbf{z}_2$ is a scalar product in Z and $k : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ is *a Mercer kernel* such that

$$\Phi(\mathbf{x}_1) \cdot \Phi(\mathbf{x}_2) = k(\mathbf{x}_1, \mathbf{x}_2) \text{ for all } \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^N.$$

Conceptually, we construct an optimal hyperplane in the features space Z for the transformed training data

$$(\mathbf{z}_i, y_i) := (\Phi(\mathbf{x}_i), y_i) \in Z \times \{-1, 1\} \text{ for } i = 1, \ldots, m.$$

Formally, in order to adapt the above algorithms to the current situation, we need to substitute each vector $\mathbf{x}_i \in \mathbb{R}^N$ by its image $\Phi(\mathbf{x}_i)$ and then to use the Mercer kernel

Mercer kernel

in order to calculate the dot products. In the typical case of very high dimensionality of Z the support centers $\mathbf{w}^{(+1)}$ and $\mathbf{w}^{(-1)}$ cannot be stored directly in the computer and they should be represented by a vector of Lagrange multipliers $\vec{\alpha} = (\alpha_i) \in \mathbf{A}$ such that

$$\mathbf{w}^{(+1)} = \sum_{i \in I^{(+1)}} \alpha_i \Phi(\mathbf{x}_i) \text{ and } \mathbf{w}^{(-1)} = \sum_{i \in I^{(-1)}}^{m} \alpha_i \Phi(\mathbf{x}_i).$$

All algebraic expressions required for the training algorithms can be derived from four different scalar products which can be recalculated after each update. We shall introduce a special notation for them now.

$$\mathbf{xw}(i, y) := \Phi(\mathbf{x}_i) \cdot \mathbf{w}^{(y)} = \sum_{j \in I^{(y)}} \alpha_j k(\mathbf{x}_j, \mathbf{x}_i) \text{ for all } i = 1, \ldots, m \tag{6.59}$$

$$\mathbf{x2}(i) := \|\Phi(\mathbf{x}_i)\|^2 = k(\mathbf{x}_i, \mathbf{x}_i) \text{ for all } i = 1, \ldots, m \tag{6.60}$$

$$\mathbf{ww}(y', y'') := \mathbf{w}^{(y)} \cdot \mathbf{w}^{(y')} = \sum_{i,j \in I^{(y)} \times I^{(y')}}^{m} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \text{ for } = y, y' = \pm 1. \tag{6.61}$$

In particular, in terms of those quantities we have the following expressions

$$\|\mathbf{w}\| = \sqrt{\mathbf{ww}(1,1) + \mathbf{ww}(-1,-1) - 2\mathbf{ww}(-1,1)}, \tag{6.62}$$

$$\begin{aligned} G(\vec{\alpha}; j) &= -y_j(\Phi(\mathbf{x}_j) - \mathbf{w}^{(y_j)}) \cdot \mathbf{w} \\ &= -y_j \left(\mathbf{xw}(j,1) - \mathbf{xw}(j,-1) - \mathbf{ww}(y_j,1) + \mathbf{ww}(y_j,-1)\right), \end{aligned} \tag{6.63}$$

$$H(\vec{\alpha}; j) = ||\mathbf{z}_j - \mathbf{w}^{(y_j)}|| = \sqrt{\mathbf{x2}(j) + \mathbf{ww}(y_j, y_j) - 2\,\mathbf{xw}(j, y_j)}, \tag{6.64}$$

for $j = 1, \ldots, m$. The trigger functions θ_{Incr} and θ_{Decr} have the same forms as before (cf. Eqns. 6.25 and 6.26), but H and G used in them should be calculated by the above formulae rather than Eqns. (6.16) and (6.17).

For the separation margins of a functional $\pi_{\mathbf{w}^{(+1)}, \mathbf{w}^{(-1)}}$ and the maximal margin we have the following formulae:

$$\rho(\pi_{\mathbf{w}^{(+1)}, \mathbf{w}^{(-1)}}) = \frac{\|\mathbf{w}\|}{2} - \max_{i=1,\ldots,m} \frac{G(\vec{\alpha}, i)}{\|\mathbf{w}\|},$$

$$\rho = \max_{(\mathbf{w}, b) \in (Z-0) \times \mathbb{R}} \frac{y_i(\mathbf{w} \cdot \Phi(\mathbf{x}) + b)}{\|\mathbf{w}\|}.$$

kernel case

All algorithms of the previous section have straightforward extensions to the kernel case and so have the convergence Theorems 6.8-6.15, where the data diameter of the transformed data set is expressed as:

$$D = \max_{1 \le i < j \le m} \sqrt{k(\mathbf{x}_i, \mathbf{x}_i) + k(\mathbf{x}_j, \mathbf{x}_j) - 2\,k(\mathbf{x}_i, \mathbf{x}_j)} \le 2 \max_{i=1,\ldots,m} \sqrt{k(\mathbf{x}_i, \mathbf{x}_i)}. \tag{6.65}$$

In these algorithms the main difference is that G and H should be calculated by the formulae (6.63) and (6.64), rather than (6.16) and (6.17).

One issue, which is not clear, is the computational complexity of the algorithms, especially in the case when vectors $\mathbf{w}^{(+1)}$ and $\mathbf{w}^{(-1)}$ cannot be represented explicitly. It can be demonstrated that with some care this is not a serious problem, since

the complexity is comparable with that of the linear (explicit $\mathbf{w}$) case. Further, it can be shown that in the kernel case the major computational cost is hidden in the calculation of kernel values for selected support vectors, and that the additional cost connected with search for the best candidate for upgrade in the case of greedy algorithms is negligible, so it pays to introduce some sophistication in this regard (cf. the discussion of computational cost in Section 6.7).

6.5 Soft Margin Extension

Now let us consider the situation when data $(\mathbf{z}_i, y_i) = (\Phi(\mathbf{x}_i), y_i)$, $i = 1, \ldots, m$, is not separable. In this section we consider the "soft margin" *support vector machine* with violations and quadratic penalty (*SVM-VQ*, following the notation in [Keerthi et al., 1999]). In feature space it takes the form of the solution to the following optimization task:

SVM-VQ

$$\begin{aligned} \mu^2 := \min_{\mathbf{w},\xi_i,b} \quad & \left(\frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{2}\sum_{i=1}^{m} \xi_i^2 \right) \\ \text{such that} \quad & y_i(\mathbf{w}\cdot\mathbf{z}_i + b) \geq 1 - \xi_i \quad (i = 1, \ldots, m). \end{aligned} \tag{6.66}$$

The significance of this formulation is that it is equivalent to the "separable" problem

SVM-NV

$$\begin{aligned} \tilde{\mu}^2 := \min_{\tilde{\mathbf{w}},b} \quad & \frac{1}{2}\|\tilde{\mathbf{w}}\|^2 \\ \text{such that} \quad & y_i(\tilde{\mathbf{w}}\cdot\tilde{\mathbf{z}}_i + \tilde{b}) \geq 1 \quad (i = 1, \ldots, m). \end{aligned} \tag{6.67}$$

under the transformation[2]:

$$\tilde{\mathbf{w}} := \left(\mathbf{w}, \sqrt{C}\vec{\xi}\right) \text{ and } \tilde{b} := b, \tag{6.68}$$

$$\tilde{\mathbf{z}}_i = \Psi_C(\mathbf{z}_i) := \left(\mathbf{z}_i, \frac{y_i}{\sqrt{C}}\vec{e}_i\right) \text{ for all } i = 1, \ldots, m, \tag{6.69}$$

where $\vec{\xi} = (\xi_i) \in \mathbb{R}^m$ is the vector of slack variables and $\vec{e}_i \in \mathbb{R}^m$ has all coordinates set to 0 with the exception of the i-th coordinate being 1. Indeed a straightforward check shows that data $(\tilde{\mathbf{z}}_i, y_i)$, $i = 1, \ldots, m$, is separable with margin $\geq (mC)^{-0.5}$ (e.g., by $\pi_{\tilde{\mathbf{w}},0}$ where $\tilde{\mathbf{w}} := (0, \sum_{i=1}^{m} \vec{e}_i) \in Z \times \mathbb{R}^m$) and $\mathbf{w}, b, \vec{\xi}$ solves SVM-VQ iff $\tilde{\mathbf{w}}, \tilde{b}$ solves (6.67). Following Keerthi et al. [1999], the latter optimization problem will be referred to as *SVM-NV*, where "NV" stands for "non-violations." The above

2. Keerthi et al. [1999] credit Frieß [1999] for this transformation and the above equivalence. However, the transformation can be traced back to the earlier research by Cristianini and Shawe-Taylor (cf. Chapter 19 of this book for details) and the equivalence was known even earlier [Cortes and Vapnik, 1995, Equation 67].

transformations ensure that

$$|\mu| = |\tilde{\mu}|. \tag{6.70}$$

It is well known that the solution $\hat{\mathbf{w}}, \hat{b}$ of (6.67) defines an optimal functional $\hat{\pi}(\tilde{\mathbf{z}}) := \hat{\mathbf{w}} \cdot \tilde{\mathbf{z}} + \hat{b}$ on $Z \times \mathbb{R}^m$ for the separable data $(\tilde{\mathbf{z}}_i, y_i)$, $i = 1, .., m$. This functional is related to the optimal functional $\pi_{\tilde{\mathbf{w}}_*^{(+1)}, \tilde{\mathbf{w}}_*^{(-1)}}$ defined by (6.7) for a pair of points $(\tilde{\mathbf{w}}_*^{(+1)}, \tilde{\mathbf{w}}_*^{(-1)})$ of the closest distance between $\operatorname{co}\{\tilde{\mathbf{z}}_i \,;\, y_i = 1\}$ and $\operatorname{co}\{\tilde{\mathbf{z}}_i \,;\, y_i = -1\}$ as follows:

$$\hat{\mathbf{w}} = \frac{\tilde{\mathbf{w}}_*^{(+1)} - \tilde{\mathbf{w}}_*^{(-1)}}{2\tilde{\rho}^2} \text{ and } \hat{b} = -\frac{\left\|\mathbf{w}_*^{(+1)}\right\|^2 - \left\|\mathbf{w}_*^{(-1)}\right\|^2}{2\tilde{\rho}^2},$$

where $\tilde{\rho} = 0.5\left\|\tilde{\mathbf{w}}_*^{(+1)} - \tilde{\mathbf{w}}_*^{(-1)}\right\|$ denotes the maximal margin for $(\tilde{\mathbf{z}}_i, y_i)$, $i = 1, \ldots, m$. (For a justification of the above equations note that $y_i \hat{\pi}(\tilde{\mathbf{z}}_i) = 1$ and $y_i \pi_{\tilde{\mathbf{w}}_*^{(+1)}, \tilde{\mathbf{w}}_*^{(-1)}}(\tilde{\mathbf{z}}_i) = 0.5\left\|\tilde{\mathbf{w}}_*^{(+1)} - \tilde{\mathbf{w}}_*^{(-1)}\right\|^2 = 2\tilde{\rho}^2$ for every support vector $\tilde{\mathbf{z}}_i$.) Hence the algorithms defined in previous sections can be used for the solution of SVM-VQ problem. It is not hard to observe that if k is a Mercer kernel corresponding to the feature transformation $\mathbb{R}^N \to Z$, $\mathbf{z}_i = \Phi(\mathbf{x}_i)$, then the Mercer kernel

$$\tilde{k}(\mathbf{x}_i, \mathbf{x}_j) := k(\mathbf{x}_i, \mathbf{x}_j) + C^{-1}\delta_{ij} \text{ with } 1 \le i, j \le m, \tag{6.71}$$

where δ_{ij} is the Kronecker delta function, is the Mercer kernel corresponding to the feature transformation $\Psi_C : \mathbb{R}^N \to Z \times \mathbb{R}^m$ (cf. Eqn. 6.69).

Hence, the task of solving an SVM-VQ for a kernel k formally reduces to an application of the kernel extension of one of Algorithms of Section 6.2 with the modified kernel (6.71). Combining all these observations with theorems of Section 6.2 we obtain a number of algorithms and theorems stating that an approximation of optimal SVM-VQ machine of predefined precision can be found in a finite number of iterations. In particular, from Theorem 6.12 and an observation that $\tilde{\rho} = \frac{1}{\sqrt{2}\tilde{\mu}} = \frac{1}{\sqrt{2}\mu}$ we obtain the following result.

Greedy MMP: convergence for SVM-VQ

Theorem 6.17

Let $0 < \epsilon < 1$, $C > 0$, k be a Mercer kernel on $\mathbb{R}^N$ and $\tilde{k}$ be its extension (6.71). The kernel extension of Algorithm 6.3 (cf. Section 6.5) applied to data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, m$, with kernel $\tilde{k}$ halts after

$$t_{\text{upd}} \le \frac{4\tilde{D}^2\mu^2}{\epsilon^2} \ln \frac{\tilde{D}\mu}{\sqrt{2}} \le \frac{2m\tilde{D}^2 C}{\epsilon^2} \ln \frac{\tilde{D}\sqrt{mC}}{2} \tag{6.72}$$

updates of $\vec{\alpha}_t$, where $\mu > 0$ is the square root of the optimal value of the functional defined in (6.66) and

$$\begin{aligned}\tilde{D} &:= \max_{i \ne j} \sqrt{k(\mathbf{x}_i, \mathbf{x}_i) + k(\mathbf{x}_j, \mathbf{x}_j) - 2k(\mathbf{x}_i, \mathbf{x}_j) + 2C^{-\frac{1}{2}}} \\ &\le 2 \max_i \sqrt{k(\mathbf{x}_i, \mathbf{x}_i) + C^{-\frac{1}{2}}}.\end{aligned}$$

Upon exit the algorithm yields a vector of Lagrange multipliers $\vec{\alpha} \in \mathbf{A}$ for which exist constants b and $E > 0$ such that $\mathbf{w} := \sum_i \alpha_i y_i \Phi(\mathbf{x}_i)/E$ and $\vec{\xi} := \vec{\alpha}/(EC)$ satisfy the following conditions (near optimal solution of the SVM-VQ):

$$\mu^2 \leq \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{2}\sum_{i=1}^{m}\xi_k^2 \leq \frac{\mu^2}{(1-\epsilon)^2} \tag{6.73}$$
$$\text{such that } y_i(\mathbf{w}\cdot\mathbf{z}_i + b) \geq 1 - \xi_i \quad (i = 1, \ldots, m).$$

Proof From 6.70 it follows that for the maximal separation margin $\tilde{\rho}$ of $(\tilde{\mathbf{z}}_i, y_i)$, $i = 1, \ldots, m$, we have

$$\tilde{\rho} = \frac{1}{\sqrt{2}\tilde{\mu}} = \frac{1}{\sqrt{2}\mu} \geq \frac{1}{\sqrt{mC}}. \tag{6.74}$$

(For the latter bound note that for the vector $\tilde{\mathbf{w}} = (0, (1, \ldots, 1)) \in Z \times \mathbb{R}^m$ we have $\tilde{\mathbf{w}}\cdot\tilde{\mathbf{z}}_i = 1/\sqrt{C}$ for all i, and that $\|\tilde{\mathbf{w}}\| = \sqrt{m}$.)

From the bound (6.47) of Theorem 6.12 it follows that Algorithm 6.3 halts after the number of updates not larger than

$$\frac{2\tilde{D}^2}{\epsilon^2\tilde{\rho}^2}\ln\frac{\tilde{D}}{2\tilde{\rho}} = \frac{4\tilde{D}^2\mu^2}{\epsilon^2}\ln\frac{\tilde{D}\mu}{\sqrt{2}} \leq \frac{2D^2mC}{\epsilon^2}\ln\frac{\tilde{D}\sqrt{mC}}{2}.$$

This completes the proof of (6.72).

Now we show that vector $\vec{\alpha} \in \mathbf{A}$ of Lagrange multipliers obtained on exit from the algorithm has the postulated properties. From Theorem 6.12 we know that

$$\tilde{\rho} \geq \rho(\pi_{\tilde{\mathbf{w}}_{\vec{\alpha}}, \tilde{b}_{\vec{\alpha}}}) \geq \tilde{\rho}(1-\epsilon), \tag{6.75}$$

where

$$\tilde{\mathbf{w}}_{\vec{\alpha}} := \sum_{i=1}^{m}\alpha_i y_i \tilde{\mathbf{z}}_i = \left(\sum_{i=1}^{m}\alpha_i y_i \mathbf{z}_i, \frac{\vec{\alpha}}{\sqrt{C}}\right) \in Z \times \mathbb{R}^m, \tag{6.76}$$

$$\begin{aligned}\tilde{b}_{\vec{\alpha}} &:= -\frac{\left\|\sum_{i\in I^{(+1)}}\alpha_i\tilde{\mathbf{z}}_i\right\|^2 - \left\|\sum_{i\in I^{(-1)}}\alpha_i\tilde{\mathbf{z}}_i\right\|^2}{2} \\ &= \frac{\left\|\sum_{i,j\in I^{(-1)}}\alpha_i\alpha_j k(\mathbf{x}_i,\mathbf{x}_j)\right\| - \left\|\sum_{i,j\in I^{(+1)}}\alpha_i\alpha_j k(\mathbf{x}_i,\mathbf{x}_j)\right\|}{2} \\ &\quad + \frac{\#I^{(-1)} - \#I^{(+1)}}{2C}.\end{aligned} \tag{6.77}$$

From the definition of the margin we have

$$y_i(\tilde{\mathbf{w}}_{\vec{\alpha}}\cdot\tilde{\mathbf{z}}_i + \tilde{b}_{\vec{\alpha}}) = y_i\pi_{\tilde{\mathbf{w}}_{\vec{\alpha}}, \tilde{b}_{\vec{\alpha}}}(\tilde{\mathbf{z}}_i) \geq \|\tilde{\mathbf{w}}_{\vec{\alpha}}\|\rho(\pi_{\tilde{\mathbf{w}}_{\vec{\alpha}}, \tilde{b}_{\vec{\alpha}}}) \text{ for all } i = 1, \ldots, m. \tag{6.78}$$

Introducing the notation

$$E := \|\tilde{\mathbf{w}}_{\vec{\alpha}}\|\rho(\pi_{\tilde{\mathbf{w}}_{\vec{\alpha}}, \tilde{b}_{\vec{\alpha}}}) > 0, \tag{6.79}$$

$$b := \tilde{b}_{\vec{\alpha}}/E, \tag{6.80}$$

we obtain from (6.75) and (6.78)

$$\frac{1}{\tilde{\rho}^2} \leq \frac{1}{\rho(\pi_{\tilde{\mathbf{w}}_{\vec{\alpha}},\tilde{b}_{\vec{\alpha}}})^2} = \left\| \frac{\tilde{\mathbf{w}}_{\vec{\alpha}}}{E} \right\|^2 \leq \frac{1}{\tilde{\rho}^2(1-\epsilon)^2}, \tag{6.81}$$

$$1 \leq \quad y_i\left(\frac{\tilde{\mathbf{w}}_{\vec{\alpha}}}{E} \cdot \tilde{\mathbf{z}}_i + b\right) \qquad \text{for all } i = 1, \ldots, m. \tag{6.82}$$

For $\mathbf{w} := \sum_{i=1}^m \alpha_i y_i \mathbf{z}_i / E \in Z$ and $\vec{\xi} := \vec{\alpha}/(EC) \in \mathbb{R}^m$ we have $\tilde{\mathbf{w}}_{\vec{\alpha}}/E = (\mathbf{w}, \sqrt{C}\vec{\xi})$. Using those $\mathbf{w}$ and $\vec{\xi}$ in Eqns. 6.81-6.82, and substituting for $\tilde{\rho}$ from (6.74), we obtain

$$\begin{aligned} \tilde{\mu}^2 \leq \quad & \frac{1}{2}(\|\tilde{\mathbf{w}}\|^2 + C\vec{\xi}^2) \quad \leq \frac{\tilde{\mu}^2}{(1-\epsilon)^2}, \\ 1 \leq\ & y_i(\tilde{\mathbf{w}} \cdot \tilde{\mathbf{z}}_i + y_i\xi_i + b) \text{ for all } i = 1, \ldots, m, \end{aligned} \tag{6.83}$$

which proves (6.73). ■

Remark 6.18
For completeness we explicitly describe the SVM machine $f_{\vec{\alpha}} : \mathbb{R}^N \to \mathbb{R}$ corresponding to the solution of (6.73) constructed in the above proof. We have

$$f_{\vec{\alpha}}(\mathbf{x}) := \frac{1}{E}\left(\sum_{j=1}^m y_i \alpha_j k(\mathbf{x}_j, \mathbf{x}) + \tilde{b}_{\vec{\alpha}}\right) \text{ for all } \mathbf{x} \in \mathbb{R}^N,$$

where $E = \|\tilde{\mathbf{w}}_{\vec{\alpha}}\| \rho(\pi_{\tilde{\mathbf{w}}_{\vec{\alpha}},\tilde{b}_{\vec{\alpha}}})$,

$$\|\tilde{\mathbf{w}}_{\vec{\alpha}}\|^2 = \frac{\|\vec{\alpha}\|^2}{C} + \sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j),$$

$$\tilde{b}_{\vec{\alpha}} = -\frac{1}{2}\sum_{y=\pm 1} y \sum_{i,j \in I^{(y)}} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + \frac{\#I^{(-1)} - \#I^{(+1)}}{2C},$$

$$\rho(\pi_{\tilde{\mathbf{w}}_{\vec{\alpha}},\tilde{b}_{\vec{\alpha}}}) = \|\tilde{\mathbf{w}}_{\vec{\alpha}}\|^{-1}\left(\tilde{b}_{\vec{\alpha}} + \min_i\left(\sum_{j=1}^m y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) + \frac{\alpha_i y_i}{C}\right)\right),$$

where "$\#I^{(y)}$" denotes cardinality of the set $I^{(y)}$.

6.6 Experimental Results

6.6.1 NIST Digits

In this section we present some initial results of a test of the kernel version of Algorithm 6.3 (Greedy MMP) on the popular NIST benchmark data set of

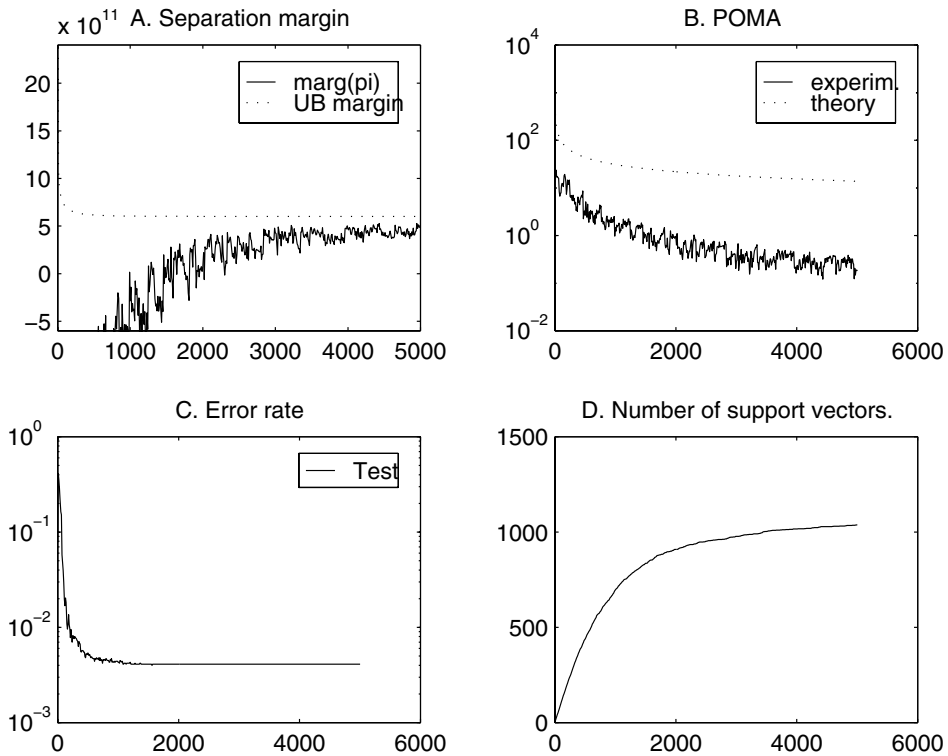


Figure 6.3 Results of the simulation of the Algorithm 6.3 on NIST hand written digits. The target was discrimination between 0 and the remaining 9 digits. Training was on 30K samples and the test on 10K samples from different writers. The horizontal axes show the number of updates t. In Figure A $marg(pi) := \rho(\pi_{\vec{\alpha}_t})$ and $\|\mathbf{w}_t\|/2$ is used as the upper bound on the margin. In Figure B we plot the experimental estimate of the precision of margin approximation (POMA) and its upper bound given by Corollary 6.13. In the expressions $\epsilon(\vec{\alpha}_t) = (\rho - \rho(\vec{\alpha}_t))/\rho$ for the experimental curve and $D\rho^{-1}\sqrt{2t^{-1}\ln(0.5D\rho^{-1})}$ used for the theoretical upper bound we have used $D := 2\sqrt{\max k(\mathbf{x}_i, \mathbf{x}_i)}$ and the final value of $\|\mathbf{w}_t\|/2$ as the substitution for ρ.

Matlab simulation

handwritten digits.[3] The software has been written in Matlab. This has some drawbacks, such as heavy requirements for memory (RAM) forcing us to use half rather than the full training set of 60K NIST digits. This restriction should disappear with the new version of Matlab, which allows storing integer data in a more efficient format than "double precision." For this test the basic logic of Algorithm 6.3 has been implemented with small enhancements, in particular, with caching values $k(\mathbf{x}_i, \mathbf{x}_j)$, $j = 1, \ldots, m$, for some frequently used patterns $\mathbf{x}_i$.

3. Data set available from http://www.research.att.com/~yann/ocr/mnist/index.html

digit recognition experiment

In Figures 6.3 we give a sample of results obtained. The target task here was the discrimination between 0 and the remaining 9 digits. The training was on 30K samples, and the test on the standard set 10K samples from different writers. The fourth degree polynomial kernel was used in this experiment. The fully trained network used 1038 support vectors, made 41 errors on the test set and achieved $> 80\%$ of the optimal margin.

comparison with experiment

It can be observed that the main progress in terms of reduction of test error rate was made in the initial stages of training, especially up to the moment of separation of the training data. After that we entered into a long process of improving the margin (up to 80% of the optimal margin), with a small improvement of the test error rate. Such behaviour was also observed in a number of similar experiments conducted so far with this data set for other kernels.

6.6.2 Benchmark on Checkers, Adult, Wisconsin Breast Cancer and Two Spirals Data

In this section we discuss a comparison of our Greedy MMP algorithm (*MMP*) with six other iterative methods of generating *support vector machines (SVM)*. These tests follow strictly the methodology used in [Keerthi et al., 1999]: we have used the same data sets which were made available to us by S. S. Keerthi, the same Gaussian kernels and the same values of constants C for the soft margin classifiers (SVM_VQ as described in Section 6.5).

The results for the six methods other than MMP were taken directly from Keerthi et al. [1999]. These were:

- the original Platt's *sequential minimal optimization (SMO)* algorithm [Platt, 1999] and its application to solving SNM_VQ task (*SMO_Q*);
- two algorithms introduced in [Keerthi et al., 1999], *the nearest point algorithm (NPA)* and *minimal norm algorithm (NMA)*;
- *the successive overrelaxation algorithm (SOR)* of Mangasarian and Musicant [1998] and its application to solving SVM_VQ task (*SOR_Q*). (The latter is an improved version of *kernel adatron* of Frieß et al. [1998] according to Keerthi et al. [1999].)

We have used only four data sets out of nine evaluated by Keerthi et al. [1999]. These were Adult 4, Wisconsin Breast Cancer, Checkers and Two Spirals. The first two of these sets originate from "real life data," the other two are popular "artificial" benchmarks. For two of them, Adult 4 and Checkers, the ratio of the number of updates to the number of selected support vectors is $\gg 1$ (up to 182 and 17855 for Adult 4 and Checkers, respectively). For the other two data sets this ratio is of order 1 (up to 2.7 and 17.9, for Wisconsin Breast Cancer and Two Spirals, respectively). This gives a good variety of benchmark tests.

The aim of these tests was to find the number of kernel evaluations required by each algorithm to produce approximations of the optimal soft margin SVMs for

a range of values of C with "tolerance" $\epsilon = 0.001$ (cf. the Appendix). A special simulator has been written for this purpose in C++ (cf. the Appendix for some details). The choice of the number of kernel evaluations as a figure of merit is justified by bearing in mind that it is the most numerically costly part of generating SVM for large input dimensionality N and that it is a metric independent of the particular hardware implementation of the algorithm.

The results for all six methods other than MMP, are given in [Keerthi et al., 1999] with the exception of SOR_Q (Adatron) for Adult 4 data, since this method was too slow to converge.

The results presented in the Appendix show clearly that that MMP performed very well on all these benchmarks, achieving results with the lowest number of kernel calls most of the time. However, what is more remarkable, for the high values of C (hence low margin M), when the tasks are becoming most challenging, the relative advantage of MMP increases. This is especially visible on Adult 4 and Checkers data, where the addition of a cache for some most recently evaluated values of the kernel was able to reduce number of kernel evaluations by factors exceeding 100 with respect to some other methods. Further, we have observed that even a very modest cache, saving only the five last vectors $\mathbf{v}_t = \Big(k(\mathbf{x}_{i_t}, \mathbf{x}_1), \ldots, k(\mathbf{x}_{i_t}, \mathbf{x}_m)\Big) \in \mathbb{R}^m$, has allowed us not only to curb proliferation of kernel calls for the largest Cs, but also reduce the number of kernel evaluations as C increases (cf. Figures 6.4-6.6 in the Appendix).

In Figure 6.6 we have also provided curves showing the minimal number of kernel evaluations necessary to simulate generated MMPs. We observe that using a cache of roughly 1/4 of m (the size of the data set, equal 4419 for Adult 4 and 465 for Checkers) allows us to generate MMP with the number of kernel evaluations approaching the ultimate minimum, the cost of the simulation of the generated SVM.

6.7 Discussion

comparison to Cortes and Vapnik [1995]

The results on digit recognition reported in Section 6.6.1 can be compared to experiments by Cortes and Vapnik [1995]. They have trained a more general, soft margin support vector machine, on the full 60K training set using chunking and some specialized quadratic optimization software. In the task of discrimination of 0 from other digits using the 4th degree polynomial kernels they reported only 19 errors on the test set but with a larger network of 1379 support vectors. They did not achieve a separation of the training data. All these differences warrant some further investigation which is beyond the scope of this chapter.

comparison to NPA of Keerthi et al. [1999]

The simulation results discussed in Section 6.6.2 obtained for the Greedy MMP (without cache) and by the NPA algorithm of Keerthi et al. [1999] on four benchmark sets are relatively close, though not identical. The closeness can be explained by the similarity of the basic approaches taken in both algorithms (e.g., reduction of

the solution to an approximation of the closest points between convex hulls; similar exit conditions). The difference can be attributed to different heuristics adopted to select upgrades. With an exception of the checkers data, the greedy search heuristic used by MMP was consistently better than the elaborate upgrades based on multiple points used in NPA. These issues warrant further investigation which is beyond scope of this chapter.

comparison to Gilbert [1966]

Now we discuss the relation of this chapter to [Gilbert, 1966] which presents an algorithm for approximation of the point of the convex set $K \subset \mathbb{R}^N$ closest to the origin. This problem is obviously a special case of the problem of finding two closest points of two convex sets which was tackled in this chapter, but it is not hard to show that both problems are in fact equivalent (see [Keerthi et al., 1999, Sec. 2]). A direct adaptation of Gilbert's algorithm to our situation and terminology of this chapter will produce two sequences of support centers $\mathbf{w}_t^{(-1)} \in \operatorname{co} X^{(-1)}$ and $\mathbf{w}_t^{(+1)} \in \operatorname{co} X^{(+1)}$, and associated vectors of Lagrange multipliers $\vec{\alpha}_t \in \mathbf{A}$ for $t = 1, 2, \ldots$ as follows. We start with arbitrary $\mathbf{w}_0^{(y)} \in \operatorname{co} X^{(y)}$, $y = \pm 1$. Having $\mathbf{w}_t^{(+1)}$, $\mathbf{w}_t^{(-1)}$ and $\vec{\alpha}_t$ defined, we select an index $i := \arg\max_j G(\vec{\alpha}_t; j)$ and then define $\mathbf{w}_{t+1}^{(y_i)}$ using the IncreaseStep leaving the other support center unchanged, i.e., $\mathbf{w}_{t+1}^{(-y_i)} := \mathbf{w}_t^{(-y_i)}$. As we have pointed out already, this is a special case of the basic MMP algorithm (Algorithm 6.1). Theorem 6.8 implies that the sequence $\mathbf{w}_t := \mathbf{w}_t^{(+1)} - \mathbf{w}_t^{(-1)}$ converges to the optimal vector $\mathbf{w}_*$. This convergence is also implied by [Gilbert, 1966, Theorem 3] (with some minor restrictions on initial vector $\mathbf{w}_0 = \mathbf{w}_0^{(1)} - \mathbf{w}_0^{(-1)}$). In particular this theorem provides the bound

$$\|\mathbf{w}_t - \mathbf{w}_*\| < 2Dt^{-1/2} \tag{6.84}$$

(which is an adaptation of Eqn. 5.5 in [Gilbert, 1966] to our notation). A straightforward geometrical argument translates this into the following upper bound on the precision of margin approximation (cf. Eqn. 6.46)

$$\epsilon_t := \frac{\rho - \rho(\pi_{\vec{\alpha}_t})}{\rho} \le \frac{D\|\mathbf{w}_t - \mathbf{w}_*\|}{\rho\|\mathbf{w}_t\|} < \frac{2D^2}{\rho\|\mathbf{w}_t\|t^{1/2}} \approx \frac{2D^2}{\rho^2 t^{-1/2}}. \tag{6.85}$$

The above bound on POMA is the best possible given (6.84). However, this bound is significantly weaker (in terms of constant) than the following bound provided by Corollary 6.13

comparison of bounds on POMA

$$\epsilon_t \le \frac{D}{\rho}\sqrt{\frac{2}{t}\ln\frac{D}{2\rho}},$$

since in a typical case $D \gg \rho$. [Gilbert, 1966, Example 2] implies that the convergence rate $\sim t^{-1/2}$ in the upper bound (6.84) on $\|\mathbf{w}_t - \mathbf{w}_*\|$ is the best possible in general case. Hence the rate $\sim t^{-1/2}$ in the above two bounds on ϵ_t is the best possible in general case and the only possible improvement can be in the constant factor. The experimental curve in Figure 6.3.B for NIST digits data shows that there is room for future improvements of such theoretical bounds, for some special cases at least.

limitations of [Gilbert, 1966]

Another limitation of theory in [Gilbert, 1966] is that his algorithm and theorems apply only to the sequence of data instances $i_1, \ldots, i_t, i_{t+1}, \ldots$ used for upgrades such that $i_{t+1} = \arg\min_{1 \le j \le m} G(\vec{\alpha}_t; j)$. In practice, when m is large, finding such i_{t+1} could be a major (numerical) problem which is admitted by Gilbert. In such a case, one would be forced to use the sequence satisfying the above requirement only approximately and Theorem 3 of Gilbert does not cover this case. However, our Algorithm 6.4 and Theorem 6.15, do not have such limitations: they explicitly specify criteria of "goodness" in terms of bounds guaranteeing the postulated rate of convergence. In other words, the on-line features of Theorem 6.8, which forced us to provide local criteria for evaluation of suitability of a data point to provide a "good" upgrade, have also lead us to Theorem 6.15, for batch mode of learning, giving theoretical results overcoming some limitations of [Gilbert, 1966].

comparison to [Michell et al., 1974]

It is worthwhile to emphasize, that in our algorithms we have utilized also the DecreaseStep which is not used in [Gilbert, 1966]. However, some forms of both our IncreaseStep and DecreaseStep are used in [Michell et al., 1974] and [Keerthi et al., 1999]. The major difference between our approach and those two papers is as follows. The two steps are used by us separately, while both Michell et al. [1974] and Keerthi et al. [1999] use them always in pairs. Our intuition is that such separation is computationally more efficient. This can be one of reasons why our Greedy MMP algorithm performed better than NPA or MNA algorithms of Keerthi et al. [1999] on the most benchmark tests presented in the Appendix. However, some additional investigation is required to clarify this issue.

cache impact

The dramatic decrease in the number of kernel evaluations required by Greedy MMP with cache observed for large values of C (hence small margin M) in Figure 6.6 in the Appendix can be explained as follows. For smaller values of M as the number of support vectors decreases (cf. Figures 6.4 and 6.5), Greedy MMP is able to find quickly the required support vectors, and then it spends most of the time on "fine tuning" Lagrange multipliers α_i. This process consists in repeated modifications of coefficients, most of the time iterating a relatively small number of support vectors. Thus even with a very small cache storing kernel values for the 5 last support vectors a significant savings in terms of kernel evaluations can be achieved (a factor ≈ 7). Our impression is that other algorithms can also be improved along these lines (note that both SMO and NPA are already using caches of critical variables!).

computational cost

Our preferred batch algorithm, Greedy MMP, requires a global search for each update. However, this additional overhead is negligible in some complex practical situations if some care is given to implementation. To illustrate this point we compare below two theoretical upper bounds on computational cost required to achieve separation margin $\ge (1-\epsilon)\rho$ (unfortunately details are beyond the scope of this chapter). For simplicity let us assume $m \gg 1$ and $N \gg 1$ and denote by c_{oper} an upper bound on the computational cost of an elementary calculation, such as addition, or subtraction, or multiplication, or division, or comparison of two numbers. In this case the computational cost required by our implementation of

Algorithm 6.4 (Linear MMP, non-kernel case) has the bound

$$cost \leq 10\ (Nm + o(Nm))\ \frac{D^2 F^2}{\rho^2 \epsilon^2} \ln \frac{D}{2\rho} \times c_{\text{oper}}.$$

In the case of Mercer kernel of either of two forms, $k(\mathbf{x}, \mathbf{x}') = F(\mathbf{x} \cdot \mathbf{x}')$ or $k(\mathbf{x}, \mathbf{x}') = F(||\mathbf{x} - \mathbf{x}'||_1)$, where $F(x)$ is a function with the computational cost comparable with c_{oper}, the cost of the kernel version of Algorithm 6.3 (Greedy MMP) has the bound

$$cost \leq 6\ (Nm + o(Nm))\ \frac{D^2}{\rho^2 \epsilon^2} \ln \frac{\|\mathbf{w}_o\|}{2\rho} \times c_{\text{oper}}.$$

This bound, for the algorithm solving the more complex kernel case, is in fact lower! The savings are coming from careful implementation and caching of some critical parameters (c.f. Section 6.4). Although the above two bounds are dominated by the worst case scenarios, they indicate that with some careful implementation, Greedy MMP algorithm could be very efficient even in the kernel case.

numerical efficiency vs. generalization

Let us observe that tests we have adopted from Keerthi et al. [1999] are in fact benchmarks on numerical efficiency of algorithms rather than their ability to produce good "generalization," which is the main goal for computational learning. The results in Figure 6.3 show that in the case of NIST digit data the ultimate accuracy can be achieved very early, with a relatively low margin, and further slow improvements in the margin do not have significant impact on generalization error. In this context, the benchmark tests as by Keerthi et al. [1999] which have measured the efficiency of algorithms in producing SVMs with margin within 0.1% of the optimum could be of little relevance to testing their generalization ability.

Adatron

Somewhat disappointing was the performance of the kernel adatron which was consistently the slowest to converge and even too slow to be included into the Adult 4 benchmark in [Keerthi et al., 1999] (marked as SOR_Q in the Figures 6.5-6.8). This happens in spite of promising theoretical results indicating that this algorithm has a good convergence rate (the replica calculations of Opper [1989] and Watkin et al. [1993] give the exponential rate of convergence to optimal solution in the thermodynamic limit, $\sim \exp(-t_{\text{upd}})$). This issue warrants some further investigation.

MMP $\not\equiv$ Adatron

It is worthwhile to emphasize that the kernel adatron and the maximal margin perceptron algorithms are not equivalent. The most obvious difference is in the way the modification of the Lagrange multipliers (α_i) is made in each iteration. Let us consider the case when the ith example is added to the support vectors. In such a case the kernel adatron will increase the ith Lagrange multiplier only, in accordance with the update $\mathbf{w} \leftarrow \mathbf{w} + \delta\ \mathbf{x}_i$, where δ is a certain number. In the case of maximal margin perceptron, such an update is more complicated. For instance, for the IncreaseStep, while the ith Lagrange multiplier is increased (cf. Eqn. 6.19), the other multipliers have to be decreased. This has to be done in accordance with the update $\mathbf{w}_{t+1}^{(y_i)} \leftarrow \mathbf{w}_t^{(y_i)}(1 - B) + B\ \mathbf{x}_i$, where $0 \leq B \leq 1$, since we have to always maintain the constraint of support centers being convex combinations of support

vectors. Thus the direction of incremental modification of vector $\mathbf{w}$ in both cases is quite different: for the Adatron it is parallel to $\mathbf{x}_i$, while for the maximal margin perceptron it is parallel to the difference $\mathbf{x}_i - \mathbf{w}^{(y_i)}$.

6.8 Conclusions

Novel algorithms are proposed for the approximation of the optimal hyperplane separating data with a maximal margin. Initial experiments show that they work on large data sets and work very well in comparison to other iterative algorithms for generation of support vector machines (with improvements exceeding factor 100, in the extreme cases). The proof of convergence and theoretical bounds on convergence rates to the optimal solution are presented and shown to provide significant improvements over some results published in the past. More systematic experiments are needed to evaluate the potential of these novel procedures, especially in an on-line learning mode, which was not experimentally evaluated here at all.

Acknowledgments

Special thanks to Herman Ferra for picking up numerous imperfections in the manuscript, to Pawel Kowalczyk for developing C++ simulator used in experimental evaluation, to S.S. Keerthi for provision of benchmark data and to Mario Marchard for stimulating discussions. The permission of Director, Telstra Research Laboratories to publish this material and a support of Australian Research Council (grant A49702201) are kindly acknowledged.

6.9 Appendix: Details of comparison against six other methods for iterative generation of support vector machines

In this section we present some details of comparison of a kernel version of Greedy MMP algorithm introduced in this chapter (MMP) with six iterative procedures evaluated previously by [Keerthi et al., 1999] (c.f. Section 6.6.2).

Our experiments were done strictly along the lines described in [Keerthi et al., 1999]. Due to time constraint and that for some bigger sets the results were provided for only some of the above methods, we have used only four data sets out of nine used by [Keerthi et al., 1999] and made available to us by S.S. Keerthi.[4] The Gaussian kernel (1.73) was used in solving the soft margin SVM-VQ task described in Section 6.5 for a range of values of C. Those values have been chosen by Keerthi

4. Warning! Although the data sets are originated from standard benchmark sets, they have been processed in an obscure way (e.g., only certain input variables are actually used). Thus for any fair comparison the data as used in [Keerthi et al., 1999] is required.

et al. [1999] to cover a range of values of *the margin* M equal to $2\tilde{\rho} = \frac{1}{\sqrt{2}\mu}$ in notation used in Section 6.5 (cf. Figure 6.1). The objective was to run the algorithm until the transformed data $(\tilde{\mathbf{z}}_i, y_i)$, $i = 1, \ldots, m$, is separated with "tolerance" $\epsilon = 0.001$, i.e., to generate $\vec{\alpha} \in \mathbf{A}$ such that

$$\rho(\pi_{\vec{\alpha}}) \geq (1-\epsilon)\frac{\|\tilde{\mathbf{w}}\|}{2} \geq (1-\epsilon)\tilde{\rho}$$

and the distance of each support vector $\tilde{\mathbf{z}}_i$, $\alpha_i > 0$, from the hyperplane $\pi_{\vec{\alpha}}{}^{-1}(0) \subset Z \times \mathbb{R}^m$ is not higher than $(1+\epsilon)\|\tilde{\mathbf{w}}\|/2$ (which is $\leq \tilde{\rho}\frac{1+\epsilon}{1-\epsilon}$). These conditions are easily seen to hold if $\vec{\alpha} \in \mathbf{A}$ is such that

$$G(\vec{\alpha}; i) = y_i(\pi_{\vec{\alpha}}(\mathbf{w}_{\vec{\alpha}}^{(y_i)}) - \pi_{\vec{\alpha}}(\mathbf{x}_i)) \leq \epsilon\pi_{\vec{\alpha}}(\mathbf{w}_{\vec{\alpha}}) = \epsilon\|\mathbf{w}_{\vec{\alpha}}\|^2, \quad (\forall i),$$
$$-G(\vec{\alpha}; i) = -y_i(\pi_{\vec{\alpha}}(\mathbf{w}_{\vec{\alpha}}^{(y_i)}) - \pi_{\vec{\alpha}}(\mathbf{x}_i)) \leq \epsilon\pi_{\vec{\alpha}}(\mathbf{w}_{\vec{\alpha}}) = \epsilon\|\mathbf{w}_{\vec{\alpha}}\|^2, \quad (\forall i,\ \alpha_i > 0),$$

or, equivalently, if

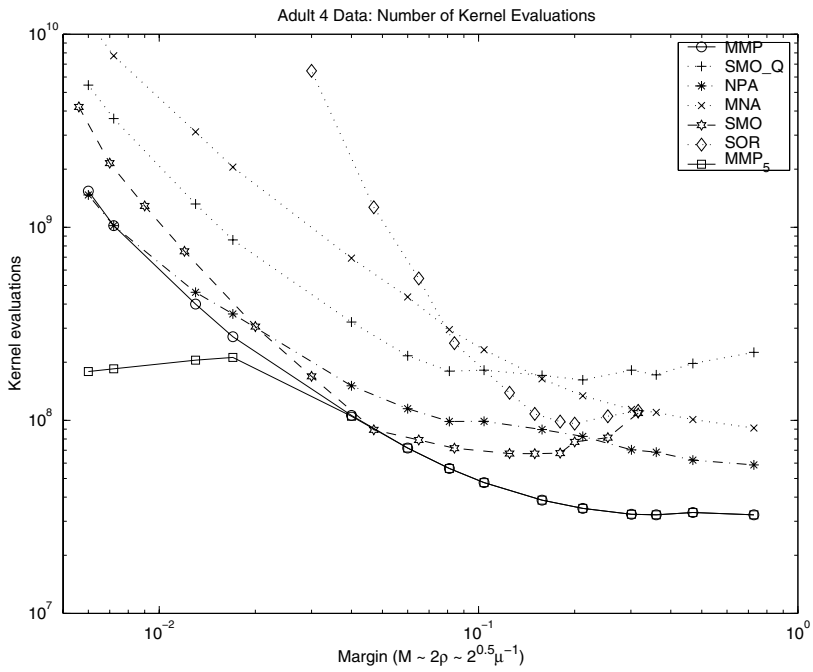
$$\frac{2\max_i G(\vec{\alpha}; i)}{\|\mathbf{w}_{\vec{\alpha}}\|^2} \leq \epsilon, \quad (\forall i),$$
$$-\frac{2\max_i G(\vec{\alpha}; i)}{\|\mathbf{w}_{\vec{\alpha}}\|^2} \leq \epsilon, \quad (\forall i,\ \alpha_i > 0).$$

The first of the above conditions is satisfied on exit from the Algorithm 6.3 but not necessarily the second one. Below we give the pseudo code of the modification of Algorithm 6.3 which we have used for these benchmark tests. Note that on exit from this algorithm the above two conditions have to be satisfied.

Algorithm 6.5 Special Greedy MMP

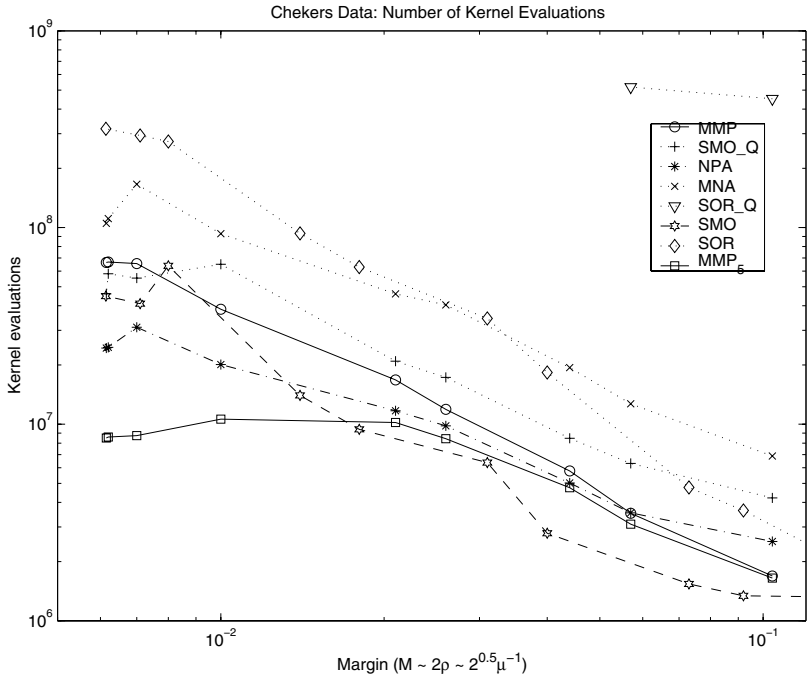
1. *Choose* ϵ, $0 < \epsilon < 1$, $\vec{\alpha}_0 \in \mathbf{A}$; $t = 1$.
2. *Repeat while* $\max_{i=1,\ldots,m} \frac{2|G(\vec{\alpha}_t, i)|}{\|\mathbf{w}_t\|^2} > \epsilon$:
   Define α_{t+1} using, respectively, the IncreaseStep (6.19) or the DecreaseStep (6.22) for i defined as follows:
   If $\max_{i=1,\ldots,m} 2G(\vec{\alpha}_t, i) > \epsilon\|\mathbf{w}_t\|^2$, then
   $i := \arg\max_j dw^2(\vec{\alpha}_t, j)$,
   else $i := \arg\max_j \max(G(j), -\chi(\alpha_i)G(j))$.
   Reset $t \leftarrow t + 1$.

The results of experiments are presented in the five figures below. Their overview has been given in Section 6.6.2.



M	C	Number of		Kernel calls $\times 10^{-7}$				
		SVs	Upd.	MMP	MMP 5	MMP 40	MMP 320	MMP 1280
0.728	0.003	4419	7319	3.24	3.24	3.24	3.24	3.24
0.469	0.1	3756	7541	3.33	3.33	3.33	3.33	3.32
0.36	0.2	3579	7332	3.24	3.24	3.24	3.24	3.19
0.301	0.3	3479	7371	3.26	3.26	3.26	3.26	3.20
0.212	0.6	3299	7924	3.50	3.50	3.50	3.50	3.38
0.158	1	3184	8730	3.86	3.86	3.86	3.86	3.63
0.104	2	3016	10769	4.76	4.76	4.76	4.75	4.04
0.081	3	2916	12743	5.63	5.63	5.63	5.60	4.25
0.06	5	2793	16277	7.19	7.19	7.18	7.03	4.34
0.04	10	2663	23906	10.56	10.52	10.47	9.82	4.65
0.017	50	2371	61211	27.05	21.21	20.28	15.50	3.23
0.013	100	2251	90583	40.03	20.48	18.83	12.24	1.78
0.0072	500	2014	231792	102.43	18.45	15.00	7.22	0.96
0.006	1000	1913	348224	153.88	17.89	14.08	5.72	0.83

Figure 6.4 Adult 4 Data: number of kernel calls, number of support vectors and the number of updates. Data for classifiers other than the maximal margin perceptron (MMP) are obtained from [Keerthi et al., 1999]. The columns "MMP$_n$" represent maximal margin perceptron algorithm with a cache preserving the n most recently calculated vectors $(k(\mathbf{x}_{i_t}, \mathbf{x}_j))_{j=1,\ldots,m} \in \mathbb{R}^m$ of Mercer kernel values (cf. Figure 6.6 for more plots).



M	C	Number of		Kernel calls $\times 10^{-6}$			
		S. vecs	Updates	MMP	MMP_5	MMP_{40}	MMP_{80}
0.104	10	245	3625	1.69	1.65	1.51	1.21
0.057	50	150	7560	3.52	3.10	2.04	1.03
0.044	100	144	12438	5.78	4.75	2.56	1.12
0.026	500	105	25589	11.9	8.43	2.20	0.078
0.021	10^3	92	36176	16.8	10.2	1.58	0.059
0.01	10^4	75	82641	38.4	10.6	0.29	0.044
0.007	10^5	75	141102	65.6	8.74	0.172	0.048
0.0062	10^6	79	143645	66.8	8.61	0.144	0.051
0.00615	10^7	80	142820	66.4	8.51	0.146	0.052

Figure 6.5 Checkers Data: number of kernel calls, number of support vectors and the number of updates. Data for classifiers other than the maximal margin perceptron (MMP) are obtained from [Keerthi et al., 1999]. MMP_n represents the maximal margin perceptron algorithm with cache preserving the n most recently calculated vectors $(k(\mathbf{x}_{i_t}, \mathbf{x}_j))_{j=1,\ldots,m} \in \mathbb{R}^m$ of Mercer kernel values (cf. Figure 6.6 for more plots).

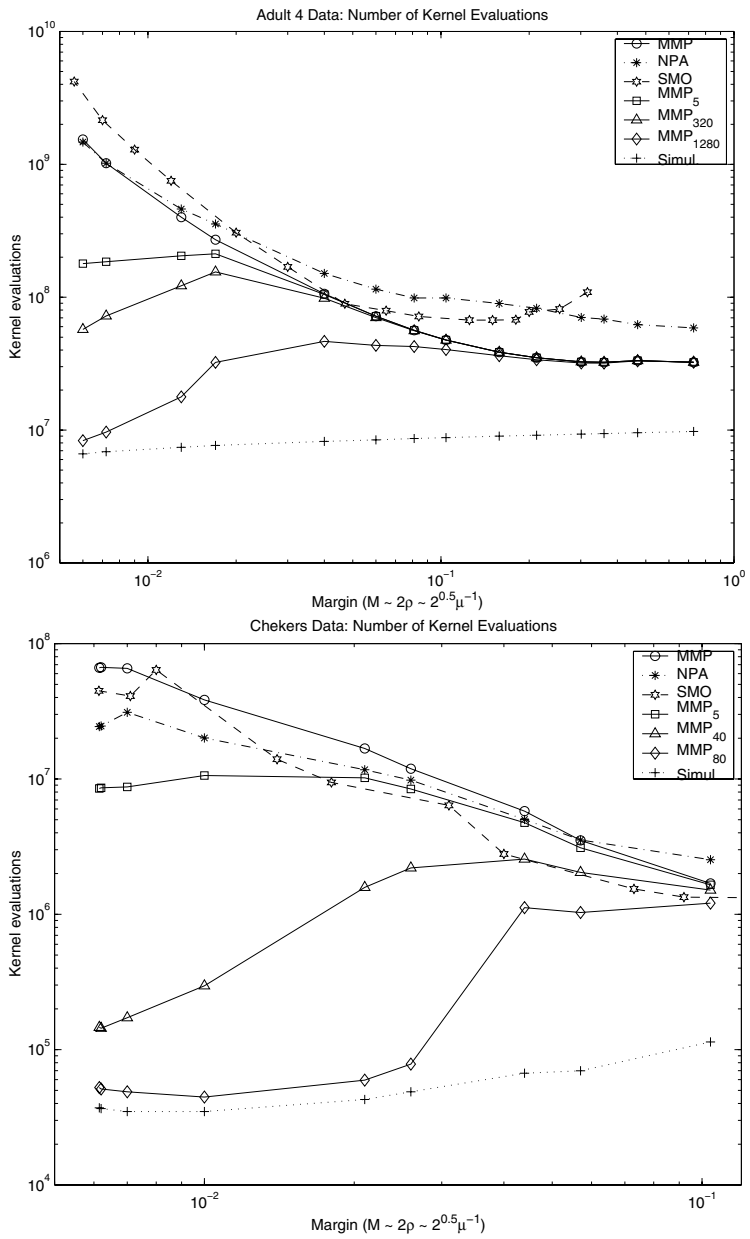
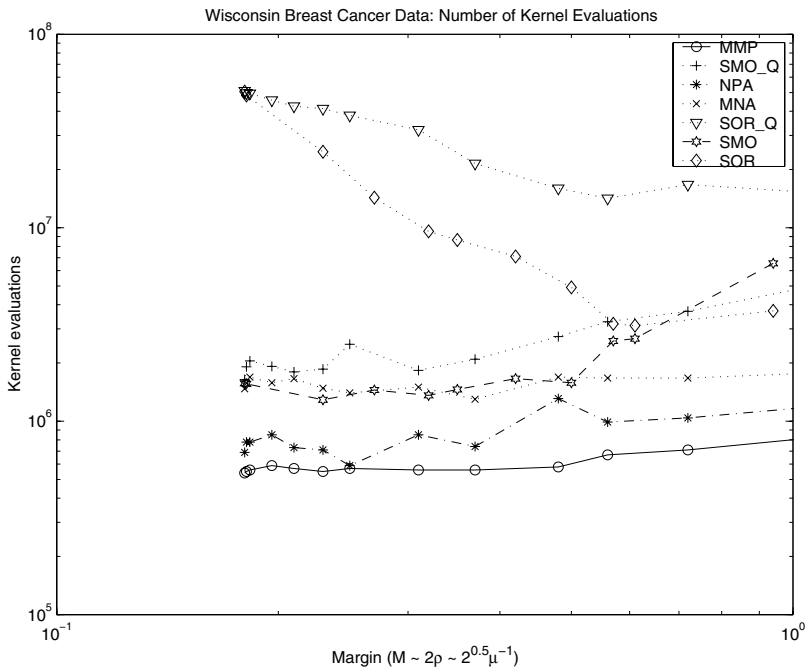
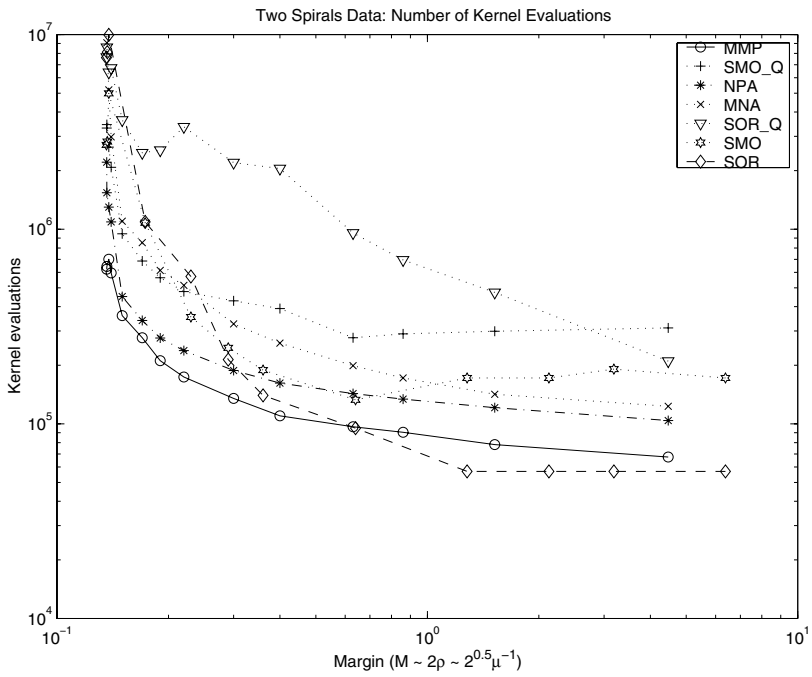


Figure 6.6 Impact of cache: number of kernel calls for the maximal margin perceptrons (MMP_n) with different cache sizes for Adult 4 and Checkers data. For clarity we have left only two best support vector machines other than MMPs from Figure 6.4 and Figure 6.5, respectively. The curves MMP_n represent maximal margin perceptron algorithm with cache preserving the n most recently calculated vectors $(k(\mathbf{x}_{i_t}, \mathbf{x}_j))_{j=1,\ldots,m} \in \mathbb{R}^m$ of Mercer kernel values. For reference we show also the curves "Simul" representing the number of kernel calls required to simulate the generated support vector machines.



M	C	Number of		Kernel calls
		Support vectors	Updates	MMP
1.13	0.03	652	1229	8.40E+05
0.72	0.1	503	1041	7.10E+05
0.56	0.2	472	981	6.70E+05
0.48	0.3	410	844	5.80E+05
0.37	0.6	361	817	5.60E+05
0.31	1	351	817	5.60E+05
0.25	2	331	830	5.70E+05
0.23	3	327	811	5.50E+05
0.21	5	317	828	5.70E+05
0.196	10	308	856	5.90E+05
0.183	50	301	818	5.60E+05
0.181	100	300	797	5.50E+05
0.18	500	296	785	5.40E+05

Figure 6.7 Wisconsin Breast Cancer Data: number of kernel calls, number of support vectors and the number of updates. Data for support vector machines other than the maximal margin perceptron (MMP) are obtained from [Keerthi et al., 1999].



M	C	Number of		Kernel calls
		Support vectors	Updates	MMP
4.47	0.03	195	345	6.75E+04
1.52	0.1	195	400	7.82E+04
0.86	0.2	195	463	9.05E+04
0.63	0.3	195	495	9.67E+04
0.4	0.6	195	564	1.10E+05
0.3	1	195	693	1.35E+05
0.22	2	195	893	1.74E+05
0.19	3	194	1079	2.11E+05
0.17	5	189	1417	2.77E+05
0.15	10	185	1845	3.60E+05
0.14	50	183	3054	5.96E+05
0.138	100	183	3596	7.01E+05
0.1364	500	179	3302	6.44E+05
0.1362	1000	178	3194	6.23E+05

Figure 6.8 Two Spirals Data: number of kernel calls, number of support vectors and the number of updates. Data for support vector machines other than the maximal margin perceptron (MMP) are obtained from [Keerthi et al., 1999].

7 Large Margin Rank Boundaries for Ordinal Regression

Ralf Herbrich
ralfh@cs.tu-berlin.de

Thore Graepel
graepel2@cs.tu-berlin.de

Klaus Obermayer
oby@cs.tu-berlin.de

Technical University of Berlin
Department of Computer Science
Franklinstr. 28/29,
10587 Berlin,
Germany

In contrast to the standard machine learning tasks of classification and metric regression we investigate the problem of predicting variables of ordinal scale, a setting referred to as *ordinal regression*. This problem arises frequently in the social sciences and in information retrieval where human preferences play a major role. Whilst approaches proposed in statistics rely on a probability model of a latent (unobserved) variable we present a distribution independent risk formulation of ordinal regression which allows us to derive a uniform convergence bound. Applying this bound we present a large margin algorithm that is based on a mapping from objects to scalar utility values thus classifying pairs of objects. We give experimental results for an information retrieval task which show that our algorithm outperforms more naive approaches to ordinal regression such as Support Vector Classification and Support Vector Regression in the case of more than two ranks.

7.1 Introduction

Let us shortly recall the model presented in Chapter 1. Given an iid sample (X, Y), and a set F of mappings $f : \mathcal{X} \mapsto \mathcal{Y}$, a learning procedure aims at finding f^* such that — using a predefined loss $c : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ — the risk functional (1.26) is minimized. Using the principle of Empirical Risk Minimization (ERM), one chooses the function f_{emp} which minimizes the mean of the loss $R_{\text{emp}}(f)$ (Equation 1.27) given the sample (X, Y). Introducing a quantity which characterizes the capacity of F, bounds for the deviation $|R(f_{\text{emp}}) - \inf_{f \in F} R(f)|$ can be derived (see Theorems 1.5, 1.6, 1.10, and 1.11). Two main scenarios were considered in the past: (i) If $\mathcal{Y}$ is a finite unordered set (nominal scale), the task is referred to as classification learning. Since $\mathcal{Y}$ is unordered, the $0-1$ loss, i.e., $c_{\text{class}}(\mathbf{x}, y, f(\mathbf{x})) = 1_{f(\mathbf{x}) \neq y}$, is adequate to capture the loss at each point $(\mathbf{x}, y)$. (ii) If $\mathcal{Y}$ is a metric space, e.g., the set of real numbers, the task is referred to as regression estimation. In this case the loss function can take into account the full metric structure. Different metric loss functions have been proposed which are optimal under given probability models $P(y|\mathbf{x})$ (cf. Huber [1981]). Usually, optimality is measured in terms of the mean squared error of f_{emp}.

classification and regression

Here, we consider a problem which shares properties of both cases (i) and (ii). Like in (i) $\mathcal{Y}$ is a finite set and like in (ii) there exists an ordering among the elements of $\mathcal{Y}$. In contrast to regression estimation we have to deal with the fact that $\mathcal{Y}$ is a non–metric space. A variable of the above type exhibits an *ordinal scale* and can be considered as the result of a coarsely measured continuous variable [Anderson and Philips, 1981]. The ordinal scale leads to problems in defining an appropriate loss function for our task (see also McCullagh [1980] and Anderson [1984]): On the one hand, there exists no metric in the space $\mathcal{Y}$, i.e., the distance $(y - y')$ of two elements is not defined. On the other hand, the simple $0-1$ loss does not reflect the ordering in $\mathcal{Y}$. Since no loss function $c(\mathbf{x}, y, f(\mathbf{x}))$ can be found that acts on true ranks y and predicted ranks $f(\mathbf{x})$, we suggest to exploit the ordinal nature of the elements of $\mathcal{Y}$ by considering the order on the space $\mathcal{X}$ induced by each mapping $f : \mathcal{X} \mapsto \mathcal{Y}$. Thus our loss function $c_{\text{pref}}(\mathbf{x}_1, \mathbf{x}_2, y_1, y_2, f(\mathbf{x}_1), f(\mathbf{x}_2))$ acts on pairs of true ranks (y_1, y_2) and predicted ranks $(f(\mathbf{x}_1), f(\mathbf{x}_2))$. Such an approach makes it possible to formulate a distribution independent theory of ordinal regression and to give uniform bounds for the risk functional. Roughly speaking, the proposed risk functional measures the probability of misclassification of a randomly drawn pair $(\mathbf{x}_1, \mathbf{x}_2)$ of observations, where the two classes are $\mathbf{x}_1 \succ_{\mathcal{X}} \mathbf{x}_2$ and $\mathbf{x}_2 \succ_{\mathcal{X}} \mathbf{x}_1$ (see Section 7.3). Problems of ordinal regression arise in many fields, e.g., in information retrieval [Wong et al., 1988, Herbrich et al., 1998], in econometric models [Tangian and Gruber, 1995, Herbrich et al., 1999b], and in classical statistics [McCullagh, 1980, Fahrmeir and Tutz, 1994, Anderson, 1984, de Moraes and Dunsmore, 1995, Keener and Waldman, 1985].

distribution independent theory of ordinal regression

As an application of the above–mentioned theory, we suggest to model ranks by intervals on the real line. Then the task is to find a latent utility function

that maps objects to scalar values. Due to the ordering of ranks, the function is restricted to be transitive and asymmetric, because these are the defining properties of a *preference relation*. The resulting learning task is also referred to as learning of preference relations (see Herbrich et al. [1998]). One might think that learning of preference relations reduces to a standard classification problem if pairs of objects are considered. This, however, is not true in general because the properties of transitivity and asymmetry may be violated by traditional Bayesian approaches due to the problem of stochastic transitivity [Suppes et al., 1989]. Considering pairs of objects, the task of learning reduces to finding a utility function that best reflects the preferences induced by the unknown distribution $p(\mathbf{x}, y)$. Our learning procedure on pairs of objects is an application of the large margin idea known from data–dependent Structural Risk Minimization [Shawe-Taylor et al., 1998]. The resulting algorithm is similar to Support Vector Machines (see Section 1.3). Since during learning and application of SVMs only inner products of object representations $\mathbf{x}_i$ and $\mathbf{x}_j$ have to be computed, the method of potential functions can be applied (see Aizerman et al. [1964] or Section 1.3.2).

preference relation

large margin

In Section 7.2 we introduce the setting of ordinal regression and shortly present well known results and models from the field of statistics. In Section 7.3 we introduce our model for ordinal regression and give a bound for the proposed loss function. In the following section we present an algorithm for ordinal regression based on large margin techniques. In Section 7.5 we give learning curves of our approach in a controlled experiment and in a real–world experiment on data from information retrieval.

7.2 Classical Models for Ordinal Regression

In this section we shortly recall the well–known cumulative or threshold model for ordinal regression [McCullagh and Nelder, 1983].

In contrast to Equation (1.2) we assume that there is an outcome space $\mathcal{Y} = \{r_1, \ldots, r_q\}$ with ordered ranks $r_q \succ_{\mathcal{Y}} r_{q-1} \succ_{\mathcal{Y}} \cdots \succ_{\mathcal{Y}} r_1$. The symbol $\succ_{\mathcal{Y}}$ denotes the ordering between different ranks and can be interpreted as "is preferred to." Since $\mathcal{Y}$ contains only a finite number of ranks, $P(y = r_i|\mathbf{x})$ is a multinomial distribution.

Let us make the assumption of stochastic ordering of the related space $\mathcal{X}$, i.e., for all different $\mathbf{x}_1$ and $\mathbf{x}_2$ either

stochastic ordering

$$\Pr(y \leq r_i|\mathbf{x}_1) \geq \Pr(y \leq r_i|\mathbf{x}_2) \qquad \text{for all } r_i \in \mathcal{Y}\,, \tag{7.1}$$

or

$$\Pr(y \leq r_i|\mathbf{x}_1) \leq \Pr(y \leq r_i|\mathbf{x}_2) \qquad \text{for all } r_i \in \mathcal{Y}\,. \tag{7.2}$$

Stochastic ordering is satisfied by a model of the form

$$l^{-1}(\Pr(y \leq r_i|\mathbf{x})) = \theta(r_i) - (\mathbf{w} \cdot \mathbf{x})\,, \tag{7.3}$$

model	inverse link function $P_\epsilon^{-1}(\Delta)$	density $dP_\epsilon(\eta)/d\eta$
logit	$\ln \frac{\Delta}{1-\Delta}$	$\frac{\exp(\eta)}{(1+\exp(\eta))^2}$
probit	$N^{-1}(\Delta)$	$\frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{\eta^2}{2}\right\}$
complementary log–log	$\ln(-\ln(1-\Delta))$	$\exp\{\eta - \exp(\eta)\}$

Table 7.1 Inverse link functions for different models for ordinal regression (taken from McCullagh and Nelder [1983]). Here, N^{-1} denotes the inverse normal function.

where $l^{-1} : [0,1] \mapsto (-\infty, +\infty)$ is a monotonic function often referred to as the inverse link function and $\theta : \mathcal{Y} \mapsto \mathbb{R}$ is increasing for increasing ranks. The stochastic ordering follows from the fact that

$$\begin{aligned}\Pr(y \leq r_i|\mathbf{x}_1) \geq \Pr(y \leq r_i|\mathbf{x}_2) &\Leftrightarrow \Pr(y \leq r_i|\mathbf{x}_1) - \Pr(y \leq r_i|\mathbf{x}_2) \geq 0 \\ &\Leftrightarrow l^{-1}(\Pr(y \leq r_i|\mathbf{x}_1)) - l^{-1}(\Pr(y \leq r_i|\mathbf{x}_2)) \geq 0 \\ &\Leftrightarrow (\mathbf{w} \cdot (\mathbf{x}_2 - \mathbf{x}_1)) \geq 0\,,\end{aligned}$$

which no longer depends on r_i (the same applies to $\Pr(y \leq r_i|\mathbf{x}_1) \leq \Pr(y \leq r_i|\mathbf{x}_2)$).

cumulative model

Such a model is called a cumulative or threshold model and can be motivated by the following argument: Let us assume that the ordinal response is a coarsely measured *latent* continuous variable $U(\mathbf{x})$. Thus, we observe rank r_i in the training set iff

$$y = r_i \Leftrightarrow U(\mathbf{x}) \in [\theta(r_{i-1}), \theta(r_i)]\,, \tag{7.4}$$

where the function U (latent utility) and $\boldsymbol{\theta} = (\theta(r_0), \ldots, \theta(r_q))^T$ are to be determined from the data. By definition $\theta(r_0) = -\infty$ and $\theta(r_q) = +\infty$. We see that the real line is divided into q consecutive intervals, where each interval corresponds to a rank r_i. Let us make a linear model of the latent variable $U(\mathbf{x})$

linear utility model

$$U(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + \epsilon\,, \tag{7.5}$$

where ϵ is the random component of zero expectation, $\mathbf{E}_\epsilon(\epsilon) = 0$, and distributed according to P_ϵ. It follows from Equation (7.4) that

$$\begin{aligned}\Pr(y \leq r_i|\mathbf{x}) &= \sum_{j=1}^{i} \Pr(y = r_j|\mathbf{x}) = \sum_{j=1}^{i} \Pr(U(\mathbf{x}) \in [\theta(r_{j-1}), \theta(r_j)]) \\ &= \Pr(U(\mathbf{x}) \in [-\infty, \theta(r_i)]) = \Pr((\mathbf{w} \cdot \mathbf{x}) + \epsilon \leq \theta(r_i)) \\ &= P(\epsilon \leq \underbrace{\theta(r_i) - (\mathbf{w} \cdot \mathbf{x})}_{\eta}) = P_\epsilon(\theta(r_i) - (\mathbf{w} \cdot \mathbf{x}))\,.\end{aligned}$$

If we now make a distributional assumption P_ϵ for ϵ we obtain the cumulative model by choosing as the inverse link function l^{-1} the inverse distribution function P_ϵ^{-1} (quantile function). Note that each quantile function $P_\epsilon^{-1} : [0,1] \mapsto (-\infty, +\infty)$ is a monotonic function. Different distributional assumptions for ϵ yield the logit, probit, or complementary log–log model (see Table 7.1).

In order to estimate $\mathbf{w}$ and $\boldsymbol{\theta}$ from model (7.3), for the observation $(\mathbf{x}_i, y)$ we see

$$\underbrace{\begin{pmatrix} o_1(\mathbf{x}_i) \\ o_2(\mathbf{x}_i) \\ \vdots \\ o_{q-2}(\mathbf{x}_i) \\ o_{q-1}(\mathbf{x}_i) \end{pmatrix}}_{\mathbf{o}(\mathbf{x}_i)} = \underbrace{\begin{pmatrix} -\mathbf{x}_i & 1 & 0 & \cdots & 0 & 0 \\ -\mathbf{x}_i & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \\ -\mathbf{x}_i & 0 & 0 & \cdots & 1 & 0 \\ -\mathbf{x}_i & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}}_{\mathbf{Z}(\mathbf{x}_i)} \underbrace{\begin{pmatrix} \mathbf{w} \\ \theta(r_1) \\ \theta(r_2) \\ \vdots \\ \theta(r_{q-2}) \\ \theta(r_{q-1}) \end{pmatrix}}_{\mathbf{w}_{\mathrm{GLM}}},$$

design matrix

where $o_j(\mathbf{x}_i) = P_\epsilon^{-1}(\Pr(y \leq r_j | \mathbf{x}_i))$ is the transformed probability of ranks less than or equal to r_j given $\mathbf{x}_i$, which will be estimated from the sample by the transformed frequencies of that event. Note that the complexity of the model is determined by the linearity assumption (7.5) and by P_ϵ^{-1} which can be thought of as a regularizer in the resulting likelihood equation. For the complete training set we obtain

$$\underbrace{\begin{pmatrix} \mathbf{o}(\mathbf{x}_1) \\ \vdots \\ \mathbf{o}(\mathbf{x}_\ell) \end{pmatrix}}_{l^{-1}(\mathbf{y}) \text{ (random)}} = \underbrace{\begin{pmatrix} \mathbf{Z}(\mathbf{x}_1) & \ldots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \ldots & \mathbf{Z}(\mathbf{x}_\ell) \end{pmatrix}}_{\mathbf{Z} \text{ (random)}} \underbrace{\begin{pmatrix} \mathbf{w}_{\mathrm{GLM}} \\ \vdots \\ \mathbf{w}_{\mathrm{GLM}} \end{pmatrix}}_{\mathbf{W}_{\mathrm{GLM}} \text{ (parameters)}} . \tag{7.6}$$

The last equation is called the design matrix of a multivariate generalized linear model (GLM). A generalized linear model $\mathbf{y} = l(\mathbf{Z}\mathbf{W}_{\mathrm{GLM}})$ is mainly determined by the design matrix $\mathbf{Z}$ and the link function $l(\cdot) = P_\epsilon(\cdot)$. Then given a sample (X, Y) and a link function — which coincides with a distributional assumption about the data — methods for calculating the maximum likelihood estimate $\mathbf{W}_{\mathrm{GLM}}$ exist (see McCullagh and Nelder [1983] or Fahrmeir and Tutz [1994] for a detailed discussion). The main difficulty in maximizing the likelihood is introduced by the nonlinear link function.

maximum likelihood estimate

To conclude this review of classical statistical methods we want to highlight the two main assumptions made for ordinal regression: (i) the assumption of stochastic ordering of the space $\mathcal{X}$ (ii) and a distributional assumption on the unobservable latent variable.

7.3 A Risk Formulation for Ordinal Regression

Instead of the distributional assumptions made in the last section, we now consider a parameterized model space G of mappings from objects to ranks. Each such function g induces an ordering $\succ_{\mathcal{X}}$ on the elements of the input space by the following rule

$$\mathbf{x}_i \succ_{\mathcal{X}} \mathbf{x}_j \Leftrightarrow g(\mathbf{x}_i) \succ_{\mathcal{Y}} g(\mathbf{x}_j) . \tag{7.7}$$

If we neglect the ordering of the space $\mathcal{Y}$, it was already shown in Section 1.1.1 that the Bayes–optimal function g^*_{class} given by Equation (1.5) is known to minimize

$$R_{\text{class}}(g) = \mathbf{E}_{\mathbf{x},y}\left(1_{g(\mathbf{x})\neq y}\right) = \mathbf{E}_{\mathbf{x},y}\left(c_{\text{class}}(\mathbf{x}, y, g(\mathbf{x}))\right) . \tag{7.8}$$

Let us rewrite $R_{\text{class}}(g)$ by

$$\begin{aligned} R_{\text{class}}(g) &= \int_{\mathcal{X}} \mathcal{Q}_{\text{class}}(\mathbf{x}, g)\; p(\mathbf{x}) d\mathbf{x} \,, \\ &\text{where} \\ \mathcal{Q}_{\text{class}}(\mathbf{x}, g) &= \sum_{i=1}^{q} \Pr(r_i|\mathbf{x}) - \Pr(g(\mathbf{x})|\mathbf{x}) = 1 - \Pr(g(\mathbf{x})|\mathbf{x}) \,. \end{aligned} \tag{7.9}$$

A closer look at Equation (7.9) shows that a sufficient condition for two mappings g_1 and g_2 to incur equal risks $R_{\text{class}}(g_1)$ and $R_{\text{class}}(g_2)$ is given by $\Pr(g_1(\mathbf{x})|\mathbf{x}) = \Pr(g_2(\mathbf{x})|\mathbf{x})$ for every $\mathbf{x}$. Assuming that $\Pr(r_i|\mathbf{x})$ is one for every $\mathbf{x}$ at a certain rank r_k the risks are equal — independently of how "far away" (in terms of rank difference) the mappings $g_1(\mathbf{x})$ and $g_2(\mathbf{x})$ are from the optimal rank $\operatorname{argmax}_{r_i\in\mathcal{Y}} \Pr(r_i|\mathbf{x})$. This evidently shows that c_{class} is inappropriate for the case where a natural ordering is defined on the elements of $\mathcal{Y}$.

Since the only available information given by the ranks is the induced ordering of the input space $\mathcal{X}$ (see Equation (7.7)) we argue that a distribution independent model of ordinal regression has to single out that function g^*_{pref} which induces the ordering of the space $\mathcal{X}$ that incurs the smallest number of inversions on pairs $(\mathbf{x}_1, \mathbf{x}_2)$ of objects (for a similar reasoning see Sobel [1993]). To model this property we note that due to the ordering of the space $\mathcal{Y}$, each mapping g induces an ordering on the space $\mathcal{X}$ by Equation (7.7). Let use define the rank difference $\ominus : \mathcal{Y}\times\mathcal{Y} \mapsto \mathbb{Z}$ by

$$r_i \ominus r_j := i - j \,. \tag{7.10}$$

Now given a pair $(\mathbf{x}_1, y_1)$ and $(\mathbf{x}_2, y_2)$ of objects we distinguish between two different events: $y_1 \ominus y_2 > 0$ and $y_1 \ominus y_2 < 0$. According to Equation (7.7) a function g violates the ordering if $y_1 \ominus y_2 > 0$ and $g(\mathbf{x}_1) \ominus g(\mathbf{x}_2) \leq 0$, or $y_1 \ominus y_2 < 0$ and $g(\mathbf{x}_1) \ominus g(\mathbf{x}_2) \geq 0$. Additionally taking into account that each weak order $\succ_{\mathcal{Y}}$ induces an equivalence $\sim_{\mathcal{Y}}$ [Fishburn, 1985] the case $y_1 \ominus y_2 = 0$ is automatically taken care of. Thus, an appropriate loss function is given by

loss function for ordinal regression

$$c_{\text{pref}}(\mathbf{x}_1, \mathbf{x}_2, y_1, y_2, g(\mathbf{x}_1), g(\mathbf{x}_2)) = \begin{cases} 1 & y_1 \ominus y_2 > 0 \wedge g(\mathbf{x}_1) \ominus g(\mathbf{x}_2) \leq 0 \\ 1 & y_2 \ominus y_1 > 0 \wedge g(\mathbf{x}_2) \ominus g(\mathbf{x}_1) \leq 0 \\ 0 & \text{else} \end{cases} \tag{7.11}$$

Note, that we can obtain m^2 samples drawn according to $p(\mathbf{x}_1, \mathbf{x}_2, y_1, y_2)$. It is important that these samples *do not provide* m^2 iid samples of the function $c_{\text{pref}}(\mathbf{x}_1, \mathbf{x}_2, y_1, y_2, g(\mathbf{x}_1), g(\mathbf{x}_2))$ for any g. Furthermore, if we define

$$c_g(\mathbf{x}_1, y_1, g(\mathbf{x}_1)) = \mathbf{E}_{\mathbf{x},y}\left[c_{\text{pref}}(\mathbf{x}_1, \mathbf{x}, y_1, y, g(\mathbf{x}_1), g(\mathbf{x}))\right] , \tag{7.12}$$

risk functional for ordinal regression

the risk functional to be minimized is given by

$$\begin{aligned} R_{\text{pref}}(g) &= \mathbf{E}_{\mathbf{x}_1,y_1,\mathbf{x}_2,y_2}\left(c_{\text{pref}}(\mathbf{x}_1,\mathbf{x}_2,y_1,y_2,g(\mathbf{x}_1),g(\mathbf{x}_2))\right) \\ &= \mathbf{E}_{\mathbf{x}_1,y_1}\left(c_g(\mathbf{x}_1,y_1,g(\mathbf{x}_1))\right) . \end{aligned} \tag{7.13}$$

Although Equation (7.13) shows great similarity to the classification learning risk functional (7.8) we see that due to the loss function c_g, which exploits the ordinal nature of $\mathcal{Y}$, we have a different pointwise loss function for each g . Thus we have found a risk functional which can be used for ordinal regression and takes into account the ordering as proposed by McCullagh and Nelder [1983].

In order to relate $R_{\text{pref}}(g)$ to a simple classification risk we slightly redefine the empirical risk based on c_{pref} and the training data (X,Y). For notational simplification let us define the space $\mathcal{E}$ of events of pairs $\mathbf{x}$ and y with unequal ranks by

$$\mathcal{E} := \{(\mathbf{z},t) \mid \mathbf{z} = (\mathbf{x}_i,\mathbf{x}_j) \in \mathcal{X}\times\mathcal{X}, t = \Omega(y_k,y_l), y_k \in \mathcal{Y}, y_l \in \mathcal{Y}, |y_k \ominus y_l| > 0\}$$

Furthermore, using the shorthand notation $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ to denote the first and second object of a pair a new training set (X',Y') can be derived from (X,Y) if we use all 2–sets in $\mathcal{E}$ derivable from (X,Y), i.e.,

$$\forall\ \ 0 < |y_i^{(1)} - y_i^{(2)}| \qquad (X',Y') = \left\{\left(\left(\mathbf{x}_i^{(1)},\mathbf{x}_i^{(2)}\right),\Omega\left(y_i^{(1)},y_i^{(2)}\right)\right)\right\}_{i=1}^{m'} \tag{7.14}$$

$$\Omega(y_1,y_2) := \operatorname{sgn}(y_1 \ominus y_2), \tag{7.15}$$

where Ω is an indicator function for rank differences and m' is the cardinality of (X',Y').

preference learning $\Leftrightarrow$ classification

Theorem 7.1 Equivalence of Risk Functionals
Assume an unknown probability measure $p(\mathbf{x},y)$ on $\mathcal{X}\times\mathcal{Y}$ is given. Then for each $g : \mathcal{X} \mapsto \mathcal{Y}$ the following equalities hold true

$$R_{\text{pref}}(g) = \mathbf{E}_{y_1,y_2}\left(|\Omega(y_1,y_2)|\right)\mathbf{E}_{\mathbf{z},t}\left(c_{\text{class}}(\mathbf{z},t,\Omega(g(\mathbf{x}_1),g(\mathbf{x}_2)))\right) , \tag{7.16}$$

$$R_{\text{emp}}(g) = \frac{m'}{m^2}\sum_{i=1}^{m'} c_{\text{class}}\left(\left(\mathbf{x}_i^{(1)},\mathbf{x}_i^{(2)}\right),\Omega\left(y_i^{(1)},y_i^{(2)}\right),\Omega\left(g\left(\mathbf{x}_i^{(1)}\right),g\left(\mathbf{x}_i^{(2)}\right)\right)\right) .$$

Proof Let us derive the probability $p(\mathbf{z},t)$ on $\mathcal{E}$ derived from $p(\mathbf{x}_1,\mathbf{x}_2,y_1,y_2)$:

$$p(\mathbf{z},t) \;=\; \begin{cases} 0 & t = 0 \\ p(\mathbf{x}_1,\mathbf{x}_2,y_1,y_2)/\Delta & t \neq 0 \end{cases} ,$$

where

$$\Delta \;=\; \mathbf{E}_{y_1,y_2}\left(|\Omega(y_1,y_2)|\right) = \Pr(|y_1 \ominus y_2| > 0) .$$

Now exploiting the definition (7.11) of c_{pref} we see

$$\forall \mathbf{x}_1,\mathbf{x}_2,y_1,y_2,g : t = c_{\text{pref}}(\mathbf{x}_1,\mathbf{x}_2,y_1,y_2,g(\mathbf{x}_1),g(\mathbf{x}_2)) .$$

The first statement is proven. The second statement follows by setting $\mathcal{X} = X, \mathcal{Y} = Y$ and assigning constant mass of $1/m^2$ at each point $(\mathbf{x}_1,\mathbf{x}_2,y_1,y_2)$. ■

Taking into account that each function $g \in G$ defines a function $p_g : \mathcal{X} \times \mathcal{X} \mapsto \{-1, 0, +1\}$ by

$$p_g(\mathbf{x}_1, \mathbf{x}_2) := \Omega(g(\mathbf{x}_1), g(\mathbf{x}_2)) \,, \tag{7.17}$$

reduction to classification problem

Theorem 7.1 states that the empirical risk of a certain mapping g on a sample (X, Y) is equivalent to the c_{class} loss of the related mapping p_g on the sample (X', Y') up to a constant factor m'/m^2 which depends neither on g nor on p_g. Thus, the problem of distribution independent ordinal regression can be reduced to a classification problem on pairs of objects. It is important to emphasize the chain of argument that lead to this equivalence. The original problem was to find a function g that maps objects to ranks given a sample (X, Y). Taking the ordinal nature of ranks into account leads to the equivalent formulation of finding a function p_g that maps pairs of objects to the three classes $\succ_{\mathcal{Y}}$, $\prec_{\mathcal{Y}}$, and $\sim_{\mathcal{Y}}$. Reverting the chain of argumentation may lead to difficulties by observing that only those p_g are admissible — in the sense that there is a function g that fulfills Equation (7.17) — which define an asymmetric, transitive relation on $\mathcal{X}$. Therefore we also call this the problem of *preference learning*. It was shown that the Bayes optimal decision function given by (1.5) on pairs of objects can result in a function p_g which is no longer transitive on $\mathcal{X}$ [Herbrich et al., 1998]. This is also known as the problem of stochastic transitivity [Suppes et al., 1989]. Note also that the conditions of transitivity and asymmetry effectively reduce the space of admissible classification functions p_g acting on pairs of objects.

uniform convergence bounds

However, the above formulation is — in the form presented — not amenable to the straightforward application of classical results from learning theory. The reason is that the constructed samples of pairs of objects violate the iid assumption. In order to still be able to give upper bounds on a risk for preference learning we have to reduce our sample such that the resulting realization of the loss (7.11) is distributed iid. Under this condition it is then possible to bound the deviation of the expected risk from the empirical risk. Let σ be any permutation of the numbers $1, \ldots, m$. Furthermore, for notational convenience let $\mathcal{C}_g(i, j)$ abbreviate $c_{\text{pref}}(\mathbf{x}_i, \mathbf{x}_j, y_i, y_j, g(\mathbf{x}_i), g(\mathbf{x}_j))$. Then we see that for any $g \in G$

$$\begin{aligned} &\Pr(\mathcal{C}_g(\sigma(1), \sigma(2)), \mathcal{C}_g(\sigma(2), \sigma(3)), \ldots, \mathcal{C}_g(\sigma(m-1), \sigma(m))) \\ &= \Pr(\mathcal{C}_g(\sigma(1), \sigma(2))) \cdot \Pr(\mathcal{C}_g(\sigma(2), \sigma(3))) \cdot \ldots \cdot \Pr(\mathcal{C}_g(\sigma(m-1), \sigma(m))) \,. \end{aligned} \tag{7.18}$$

Clearly, $m - 1$ is the maximum number of pairs of objects that still fulfil the iid assumption. In order to see this consider that by transitivity the ordering $g(\mathbf{x}_1) \prec_{\mathcal{Y}} g(\mathbf{x}_2)$ and $g(\mathbf{x}_2) \prec_{\mathcal{Y}} g(\mathbf{x}_3)$ implies $g(\mathbf{x}_1) \prec_{\mathcal{Y}} g(\mathbf{x}_3)$ (and vice versa for $\succ_{\mathcal{Y}}$ and $\sim_{\mathcal{Y}}$). Now we can give the following theorem.

Theorem 7.2 A Margin Bound for Ordinal Regression
Let p be a probability measure on $\mathcal{X} \times \{r_1, \ldots, r_q\}$, let (X, Y) be a sample of size m drawn iid from p. Let σ be any permutation of the numbers $1, \ldots, m$. For each function $g : \mathcal{X} \mapsto \{r_1, \ldots, r_q\}$ there exists a function $f \in F$ and a vector $\boldsymbol{\theta}$ such

that[1]

$$g(\mathbf{x}) = r_i \Leftrightarrow f(\mathbf{x}) \in [\theta(r_{i-1}), \theta(r_i)]\,. \tag{7.19}$$

Let the fat–shattering dimension of the set of functions F be bounded above by the function $\mathrm{afat}_F : \mathbb{R} \mapsto \mathbb{N}$. Then for each function g with zero training error, i.e., $\sum_{i=1}^{m-1} C_g(\sigma(i), \sigma(i+1)) = 0$ and

$$\rho_f = \min_{i=1,\ldots,m-1} \Omega\left(y_{\sigma(i)}, y_{\sigma(i+1)}\right) \left|f(\mathbf{x}_{\sigma(i)}) - f(\mathbf{x}_{\sigma(i+1)})\right|$$

with probability $1 - \delta$

$$R_{\mathrm{pref}}(g) \leq \frac{2}{m-1}\left(k \log_2\left(\frac{8e(m-1)}{k}\right)\log_2(32(m-1)) + \log_2\left(\frac{8(m-1)}{\delta}\right)\right),$$

where $k = \mathrm{afat}_F(\rho_f/8) \leq e(m-1)$.

Proof Let us recall the following theorem based on Theorem 1.10.

Theorem 7.3 [Shawe-Taylor et al., 1998]
Consider a real valued function class F having fat shattering function bounded above by the a function $\mathrm{afat}_F : \mathbb{R} \mapsto \mathbb{N}$ which is continuous from the right. Fix $\theta \in \mathbb{R}$. Then with probability $1 - \delta$ a learner that correctly classifies m iid generated examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ with $h = T_\theta(f) \in T_\theta(F)$ such that $h(\mathbf{x}_i) = y_i, i = 1, \ldots, m$ and $\rho_f = \min_i y_i\left(|f(\mathbf{x}_i) - \theta|\right)$ will have error of h bounded from above by

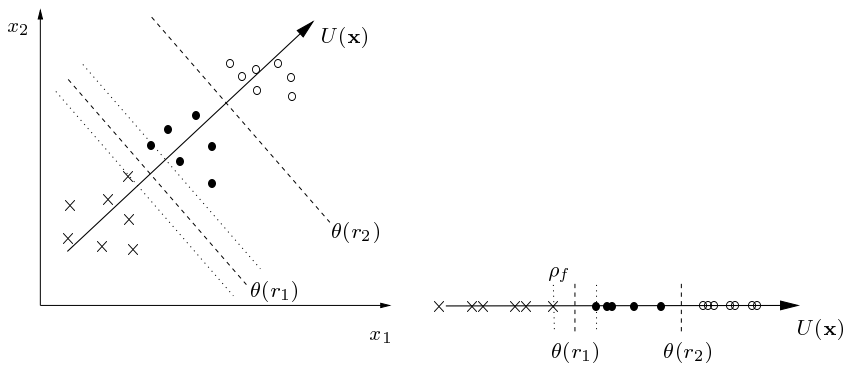
$$\frac{2}{m}\left(k \log_2\left(\frac{8em}{k}\right)\log_2(32m) + \log_2\left(\frac{8m}{\delta}\right)\right), \tag{7.20}$$

where $k = \mathrm{afat}_F(\rho_f/8) \leq em$.

Taking into account that by construction we got $m - 1$ iid examples and that the classification of a pair is carried out by a decision based on the difference $f(\mathbf{x}_{\sigma(i)}) - f(\mathbf{x}_{\sigma(i+1)})$ we can upper bound $R_{\mathrm{pref}}(g)$ by replacing each m with $m - 1$ and using $\theta = 0$. ■

The $\mathrm{afat}_F(\rho)$–shattering dimension of F can be thought of as the maximum number of objects that can be arranged in any order using functions from F and a minimum margin $\min \Omega(y_1, y_2)|f(\mathbf{x}_1) - f(\mathbf{x}_2)|$ of ρ (utilizing Equation (7.7) together with (7.19)). Note, that the zero training error condition for the above bound is automatically satisfied for any σ if $R_{\mathrm{emp}}(g) = 0$. Even though this empirical risk was not based on an iid sample its minimization allows the application of the above bound. In the following section we will present an algorithm which aims at minimizing exactly that empirical risk while at the same time enforcing large margin rank boundaries.

1. Note the close relationship to the cumulative model presented in Section 7.2.



(a) (b)

Figure 7.1 (**a**) Mapping of objects from rank r_1 ($\times$), rank r_2 ($\bullet$), and rank r_3 ($\circ$) to the axis $f(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2)^T$. Note that by $\theta(r_1)$ and $\theta(r_2)$ two coupled hyperplanes are defined. (**b**) The margin of the coupled hyperplanes $\rho_f = \min_{(X',Y')} \Omega(y_i^{(1)}, y_i^{(2)}) |f(\mathbf{x}_i^{(1)}) - f(\mathbf{x}_i^{(2)})|$ is this time defined at the rank boundaries $\theta(r_i)$.

7.4 An Algorithm for Ordinal Regression

Based on the results of Theorem 7.2 we suggest to model ranks as intervals on the real line. Similarly to the classical cumulative model used in ordinal regression, let us introduce a (latent) linear function $f : \mathcal{X} \mapsto \mathbb{R}$ for each function g

$$f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) \,, \tag{7.21}$$

which are related by (7.19). In order to apply the given theorem we see that we have to find a function f^* which incurs no training error on (X', Y') while controlling the generalization error by maximizing the margin ρ_f. Note, that

ranks as intervals on the real line

$$f(\mathbf{x}_i) - f(\mathbf{x}_j) = (\mathbf{w} \cdot (\mathbf{x}_i - \mathbf{x}_j)) \,,$$

which makes apparent that each pair $(\mathbf{x}_i, \mathbf{x}_j) \in X'$ is represented by its difference vector $(\mathbf{x}_i - \mathbf{x}_j)$ assuming a linear model of f. This allows the straightforward application of the large margin algorithm given by Equation (1.51) and (1.52) replacing each $\mathbf{x}_i$ by $(\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)})$. Hence, the maximization of the margin takes place at the rank boundaries $\theta(r_i)$ (see Equation (7.19) and Figure 7.1). In practice it is preferable to use the soft margin extension of the large margin algorithm (see Equation (1.25)). Furthermore due to the KKT conditions (see Equation (1.54)) $\mathbf{w}^*$ can be written in terms of the training data. This gives

$$\mathbf{w}^* = \sum_{i=1}^{m'} \alpha_i^* t_i \left(\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)} \right) , \tag{7.22}$$

soft margin

where $\boldsymbol{\alpha}^*$ is given by

$$\boldsymbol{\alpha}^* = \underset{\substack{C\mathbf{1}\geq\boldsymbol{\alpha}\geq\mathbf{0}\\(\boldsymbol{\alpha}\cdot\mathbf{t})=0}}{\operatorname{argmax}} \left[\sum_{i=1}^{m'}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m'}\alpha_i\alpha_j t_i t_j \left((\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)})\cdot(\mathbf{x}_j^{(1)} - \mathbf{x}_j^{(2)})\right)\right], \tag{7.23}$$

and $\mathbf{t} = (\Omega(y_1^{(1)}, y_1^{(2)}), \ldots, \Omega(y_{m'}^{(1)}, y_{m'}^{(2)}))$. Note, however, that due to the expansion of the last term in (7.23),

$$\left((\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)})\cdot(\mathbf{x}_j^{(1)} - \mathbf{x}_j^{(2)})\right) = (\mathbf{x}_i^{(1)}\cdot\mathbf{x}_j^{(1)}) - (\mathbf{x}_i^{(1)}\cdot\mathbf{x}_j^{(2)}) - (\mathbf{x}_i^{(2)}\cdot\mathbf{x}_j^{(1)}) + (\mathbf{x}_i^{(2)}\cdot\mathbf{x}_j^{(2)}),$$

the solution $\boldsymbol{\alpha}^*$ to this problem can be calculated solely in terms of the inner products between the feature vectors without reference to the feature vectors themselves. Hence, the idea of (implicitly) mapping the data X via a nonlinear mapping $\Phi : \mathcal{X} \mapsto \mathcal{F}$ into a feature space $\mathcal{F}$ can successfully applied (for further details see Section 1.3.2). Replacing each occurrence of $\mathbf{x}$ by $\Phi(\mathbf{x})$ gives

kernel trick

$$\boldsymbol{\alpha}^* = \underset{\substack{C\mathbf{1}\geq\boldsymbol{\alpha}\geq\mathbf{0}\\(\boldsymbol{\alpha}\cdot\mathbf{t})=0}}{\operatorname{argmax}} \left[\sum_{i=1}^{t}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{t}\alpha_i\alpha_j t_i t_j \mathcal{K}\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}, \mathbf{x}_j^{(1)}, \mathbf{x}_j^{(2)}\right)\right]. \tag{7.24}$$

where $\mathcal{K}$ is for a given function k defined by

$$\mathcal{K}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4) = k(\mathbf{x}_1, \mathbf{x}_3) - k(\mathbf{x}_1, \mathbf{x}_4) - k(\mathbf{x}_2, \mathbf{x}_3) + k(\mathbf{x}_2, \mathbf{x}_4). \tag{7.25}$$

Here, $k : \mathcal{X}\times\mathcal{X} \mapsto \mathbb{R}$ is a Mercer kernel and for a fixed mapping Φ is defined by

$$k(\mathbf{x}, \mathbf{x}') = (\Phi(\mathbf{x})\cdot\Phi(\mathbf{x}')).$$

Some kernels k to be used in learning are given by Equations (1.63) and (1.73). Note that the usage of kernels instead of explicitly performing the mapping Φ allows us to deal with nonlinear functions f without running into computational difficulties. Moreover, as stated in Theorem 7.2 the bound on the risk $R_{\text{pref}}(\mathbf{w})$ does not depend on the dimension of $\mathcal{F}$ but on the margin ρ_f.

In order to estimate the rank boundaries we note that due to Equations (1.52) the difference in f^* is greater or equal to one for all training examples which constitute a correctly classified pair. These can easily be obtained by checking $0 < \alpha_i^* < C$, i.e., training patterns which do not meet the box constraint (see Section 1.1.4). Thus if $\Theta(k) \subset X'$ is the fraction of objects from the training set with $0 < \alpha_i^* < C$ and rank difference $\ominus$ exactly one starting from rank r_k, i.e.,

rank boundaries

$$\Theta(k) = \left\{\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}\right) \middle| y_i^{(1)} = r_k \wedge y_i^{(2)} = r_{k+1} \wedge 0 < \alpha_i^* < C\right\} \tag{7.26}$$

then the estimation of $\theta(r_k)$ is given by

$$\theta^*(r_k) = \frac{f^*(\mathbf{x}_1) + f^*(\mathbf{x}_2)}{2}, \tag{7.27}$$

where

$$(\mathbf{x}_1, \mathbf{x}_2) = \underset{(\mathbf{x}_i, \mathbf{x}_j)\in\Theta(k)}{\operatorname{argmin}} [f^*(\mathbf{x}_i) - f^*(\mathbf{x}_j)]. \tag{7.28}$$

In other words, the optimal threshold $\theta^*(r_k)$ for rank r_k lies in the middle of the utilities of the closest (in the sense of their utility) objects of rank r_k and r_{k+1}. After the estimation of the rank boundaries $\theta(r_k)$ a new object is assigned to a rank according to Equation (7.19).

coupled hyperplanes

We want to emphasize that taking the difference vector as a representation of a pair of objects effectively couples all hyperplanes $f(\mathbf{x}) = \theta(r_k)$ thus resulting in a standard QP problem. Furthermore, the effective coupling is retained if we use general ℓ_q–margins (see Section 1.1.4). It is the reduction of the hypothesis space which makes the presented algorithm suited for the task of ordinal regression. Note, that also the kernel $\mathcal{K}$ derived from k acts only in $\mathcal{F}$ and thus avoids considering too large a hypothesis space. All properties are consequences of the *modeling of ranks* as intervals on the real line and of the prior knowledge of the ordering of $\mathcal{Y}$.

7.5 Experimental Results

In this section we present some experimental results for the algorithm presented in Section 7.4. We start by giving results for artificial data which allows us to analyze our algorithm in a controlled setting. Then we give learning curves for an example from the field of information retrieval.

7.5.1 Learning Curves for Ordinal Regression

multi-class SVM and support vector regression

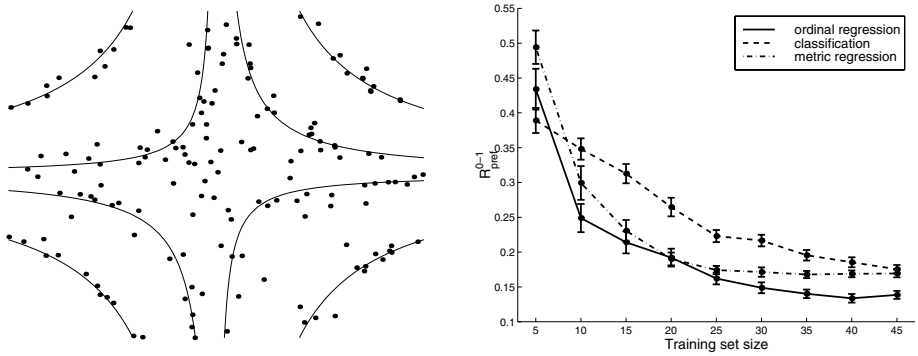
In this experiment we want to compare the generalization behavior of our algorithm with the multi-class SVM [Weston and Watkins, 1998] and Support Vector regression (SVR) (cf. Smola [1998]) — the methods of choice, if one does not pay attention to the ordinal nature of $\mathcal{Y}$ and instead treats ranks as classes (classification) or continuous response values (regression estimation). Another reason for choosing those algorithms is their similar regularizer $\|\mathbf{w}\|^2$ and hypothesis space F which make them as comparable as possible. We generated 1000 observations $\mathbf{x} = (x_1, x_2)$ in the unit square $[0,1] \times [0,1] \subset \mathbb{R}^2$ according to a uniform distribution. We assigned to each observation $\mathbf{x}$ a value y according to

$$y = i \Leftrightarrow \underbrace{10((x_1 - 0.5) \cdot (x_2 - 0.5))}_{f(\mathbf{x})} + \epsilon \in [\theta(r_{i-1}), \theta(r_i)]\,, \tag{7.29}$$

example utility function

where ϵ was normally distributed, i.e., $\epsilon \sim N(0, 0.125)$, and $\boldsymbol{\theta} = (-\infty, -1, -0.1, 0.25, 1, +\infty)$ is the vector of predefined thresholds. In Figure 7.2 (a) the points $\mathbf{x}_i$ which are assigned to a different rank after the addition of the normally distributed quantity ϵ_i are shown. If we treat the whole task as a classification problem, we would call them incorrectly classified training examples. The solid lines in Figure 7.2 (a) indicate the "true" rank boundaries $\boldsymbol{\theta}$ on $f(\mathbf{x})$.

In order to compare the three different algorithms we randomly drew 100 training samples (X, Y) of training set sizes m ranging from 5 to 45, thereby making sure



(a) (b)

Figure 7.2 (**a**) Scatter plot of data points $\mathbf{x}$ which $f(\mathbf{x})$ maps to a different interval than $f(\mathbf{x})+\epsilon$ (see Equation (7.29)). (**b**) Learning curves for multi-class SVM (dashed lines), SV regression (dashed–dotted line) and the algorithm for ordinal regression (solid line) if we measure R_{pref}. The error bars indicate the 95% confidence intervals of the estimated risk R_{pref}.

comparison to other methods

that at least one representative of each rank was within the drawn training set. Classification with multi-class SVMs was carried out by computing the pairwise $5 \cdot 4/2 = 10$ hyperplanes. For all algorithms, i.e., multi-class SVMs, SVR, and the algorithm presented in Section 7.4, we chose the kernel $k(\mathbf{x}_i, \mathbf{x}_j) = ((\mathbf{x}_i \cdot \mathbf{x}_j) + 1)^2$ and a trade-off parameter $C = 1000000$. In the particular case of Support Vector regression we used a value of $\varepsilon = 0.5$ for the ε–insensitive loss function (see [Vapnik, 1995] for the definition of this loss function) and thresholds $\boldsymbol{\theta} = (0.5, 1.5, 2.5, 3.5, 4.5)$ to transform real valued predictions into ranks.

learning curves

In order to estimate the risk $R_{\text{pref}}(g^*)/\mathbf{E}_{y_1,y_2}(|\Omega(y_1, y_2)|)$ from the remaining 995 to 955 data points we averaged over all 100 results for a given training set size. Thus we obtained the three learning curves shown in Figure 7.2 (b). Note that we used the scaled R_{pref} — which is larger by a constant factor. It can be seen that the algorithm proposed for ordinal regression generalizes much faster by exploiting the ordinal nature underlying $\mathcal{Y}$ compared to classification. This can be explained by the fact that due to the model of a latent utility all "hyperplanes" $f(\mathbf{x}) = \theta(r_k)$ are coupled (see Figure 7.1) which does not hold true for the case of multi-class SVMs. Furthermore, the learning curves for SVR and the proposed ordinal regression algorithm are very close which can be explained by the fact that the predefined thresholds $\theta(r_k)$ are defined in such a way that their pairwise difference is about 0.5 — the size of the ε–tube chosen beforehand. Thus the utility and the continuous ranks estimated by the regression algorithm are of the same magnitude which results in the same generalization behavior.

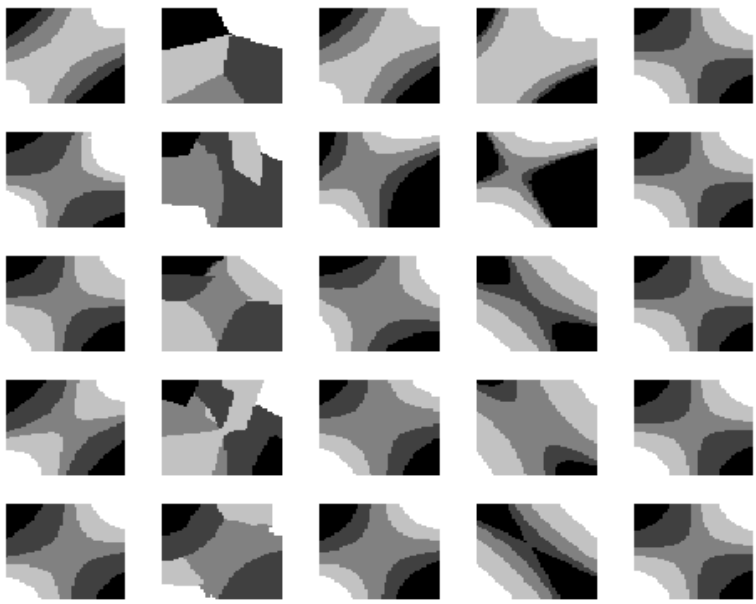
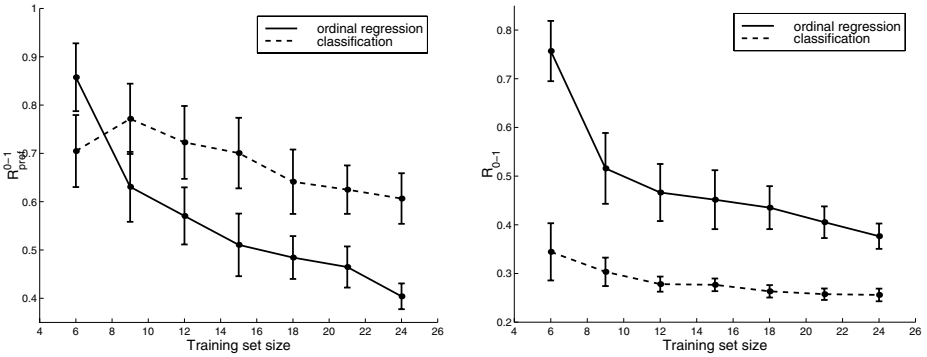


Figure 7.3 Assignments of points to ranks r_1 (black area) to r_5 (white area) by the learned function $g^*(\mathbf{x})$ based on randomly drawn training samples of size $5, 10, 15, 20,$ and 25 (top row to bottom row). (**a**) Results of the algorithm presented in Section 7.4. (**b**) Results of multi-class SVM if we treat each rank as a class. (**c**) Results of SVR if we assign rank r_i to number i. (**d**) Results of SVR if we assign rank r_i to real number $\exp(i)$. (**e**) Underlying assignment uncorrupted by noise.

In Figure 7.3 we plotted the assignments of the unit square to ranks r_1 (black areas) to ranks r_5 (white areas) for the functions $g^*(\mathbf{x})$ learned from randomly drawn training sets ranging from size $m = 5$ (top row) to $m = 25$ (bottom row). We used the same parameters as for the computation of the learning curves. In the rightmost column (e) the true assignment, i.e., $y = r_i \Leftrightarrow f(\mathbf{x}) \in [\theta(r_{i-1}), \theta(r_i)]$ is shown. In the first column (a) we can see how the algorithm presented in Section 7.4 performs for varying training set sizes. As expected, for the training set size $m = 25$, the method found a utility function together with a set of thresholds which represent the true ranking very well. The second column (b) shows the results of the abovementioned multi-class SVM on the task. Here the pairwise hyperplanes are not coupled since the ordinal nature of $\mathcal{Y}$ is not taken into account. This results in a worse generalization, especially in regions, where no training points were given. The third column (c) gives the assignments made by the SVR algorithm if we represent each rank r_i by i. Similar to the good results seen in the learning curve, the generalization behavior is comparable to the ordinal regression method (first column). The deficiency of SVR for this task becomes apparent when we change the representation of ranks. In the fourth column (d) we applied the same SVR



(a) (b)

Figure 7.4 Learning curves for multi-class SVM (dashed lines) and the algorithm for ordinal regression (solid line) for the OHSUMED dataset query 1 if we measure (**a**) R_{pref} and (**b**) R_{class}. Error bars indicate the 95% confidence intervals.

representation of ranks

algorithm, this time on the representation $\exp(i)$ for rank r_i. As can be seen, this dramatically changes the generalization behavior of the SVR method. We conclude that the crucial task for application of metric regression estimation methods to the task of ordinal regression is the definition of the representation of ranks. This is automatically — although more time–consuming — solved by the proposed algorithm.

7.5.2 An Application to Information Retrieval

information retrieval

In this experiment we make the following assumption: After an initial (textual) query a user makes to an IR system, the system returns a bundle of documents to the user. Now the user assigns ranks to a small fraction of the returned documents and the task for the learning algorithm is to assign ranks to the remaining unranked documents in order to rank the remaining documents. We assume that the quantity of interest is the percentage of inversions incurred by the ranking induced by the learning algorithm. This quantity is captured by $R_{\text{emp}}(g)/m'$ ($m' = |(X', Y')|$, see Equation (7.14) for an exact definition) and thus after using $m = 6$ up to $m = 24$ documents and their respective ranking we measure this value on the remaining documents. For this experiment we used the same parameters as in the previous experiment. The investigated dataset was the OHSUMED dataset collected by William Hersh[2], which consists of 348 566 documents and 106 queries with their respective ranked results. There are three ranks: "document is relevant," "document is partially relevant," and "irrelevant document" wrt. the given textual

2. This dataset is publicly available at `ftp://medir.ohsu.edu/pub/ohsumed/`.

query. For our experiments we used the results of query 1 ("Are there adverse effects on lipids when progesterone is given with estrogen replacement therapy?") which consists of 107 documents taken from the whole database. In order to apply our algorithm we used the bag–of–words representation [Salton, 1968], i.e., we computed for every document the vector of "term–frequencies–inverse–document–frequencies" (TFIDF) components. The TFIDF is a weighting scheme for the bag–of–words representation which gives higher weights to terms which occur very rarely in all documents. We restricted ourselves to terms that appear at least three times in the whole database. This results in $\approx$ 1700 terms which leads for a certain document to a very high–dimensional but sparse vector. We normalized the length of each document vector to unity (see Joachims [1998]).

bag–of–words representation

Figure 7.4 (a) shows the learning curves for multi-class SVMs and our algorithm for ordinal regression measured in terms of the number of incurred inversions. As can be seen from the plot, the proposed algorithm shows very good generalization behavior compared to the algorithm which treats each rank as a separate class. Figure 7.4 (b) shows the learning curves for both algorithms if we measure the number of misclassifications — treating the ranks as classes. As expected, the multi-class SVMs perform much better than our algorithm. It is important to note again, that minimizing the zero–one loss R_{class} does not automatically lead to a minimal number of inversions and thus to an optimal ordering.

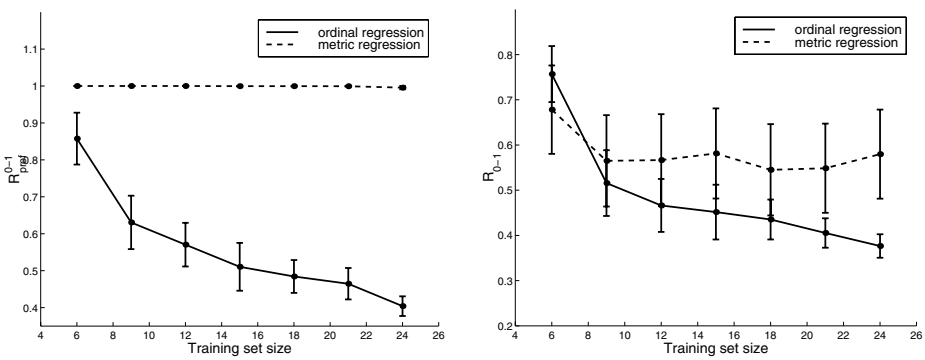


Figure 7.5 Learning curves for SVR (dashed lines) and the algorithm for ordinal regression (solid line) for the OHSUMED dataset query 1 if we measure (**a**) R_{pref} and (**b**) R_{0-1}. Error bars indicate the 95% confidence intervals.

Figure 7.5 (a) shows the learning curves for SVR and for our algorithm for ordinal regression, measured the number of incurred inversions. While the former performs quite well on the artificial dataset, in the real world dataset the SVR algorithm fails to find a ranking which minimizes the number of inversions. This can be explained by fact that for the real–world example the equidistance in the assumed utility may no longer hold — especially taking into account that the data space is very sparse

for this type of problem. Similarly, Figure 7.5 (b) shows the learning curves for both algorithms if we measure the number of misclassifications. As expected from the curves on the right the SVR algorithm is worse even on that measure. Note that the SVR algorithm minimizes neither R_{pref} nor R_{0-1} which may explain its bad generalization behavior. Also note that we made no adaptation of the parameter ε — the size of the tube. The reason is that in this particular task there would not be enough training examples available to set aside a reasonable portion of them for validation purposes.

7.6 Discussion and Conclusion

In this chapter we considered the task of ordinal regression which is mainly characterized by the ordinal nature of the outcome space $\mathcal{Y}$. All known approaches to this problem (see Section 7.2) make distributional assumptions on an underlying continuous random variable. In contrast, we proposed a loss function which allows for application of distribution independent methods to solve ordinal regression problems. By exploiting the fact that the induced loss function class is a set of indicator functions we could give a distribution independent bound on our proposed risk. Moreover, we could show that to each ordinal regression problem there exists a corresponding preference learning problem on pairs of objects. This result built the link between ordinal regression and classification methods — this time on pairs of objects. For the representation of ranks by intervals on the real line, we could give margin bounds on our proposed risk — this time applied at the rank boundaries. Based on this result we presented an algorithm which is very similar to the well known Support Vector algorithm but effectively couples the hyperplanes used for rank determination.

learning of equivalence relation

Noting that our presented loss involves pairs of objects we see that the problem of multi-class classification can also be reformulated on pairs of objects which leads to the problem of learning an *equivalence relation*. Usually, in order to extend a binary classification method to multiple classes, one–against–one or one–against–all techniques are devised [Hastie and Tibshirani, 1998, Weston and Watkins, 1998]. Such techniques increase the size of the hypothesis space quadratically or linearly in the number of classes, respectively. Recent work [Phillips, 1999] has shown that learning equivalence relations can increase the generalization behavior of binary–class methods when extended to multiple classes.

Further investigations will include the following question: does the application of the GLM methods presented in Section 7.2 lead automatically to large margins (see Theorem 7.2)? The answer to such a question would finally close the gap between methods extensively used in the past to theories developed currently in the field of Machine Learning.

Acknowledgments

First of all we are indebted to our collaborator Peter Bollmann-Sdorra who first stimulated research on this topic. We also thank Nello Cristianini, Ulrich Kockelkorn, Gerhard Tutz, and Jason Weston for fruitful discussions. Finally, we would like to thank our anonymous reviewers for very helpful comments on the uniform convergence results.

II Kernel Machines

8 Generalized Support Vector Machines

Olvi L. Mangasarian
Computer Sciences Department
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706, USA
olvi@cs.wisc.edu
http://www.cs.wisc.edu/∼olvi

By setting apart the two functions of a support vector machine: separation of points by a nonlinear surface in the original space of patterns, and maximizing the distance between separating planes in a higher dimensional space, we are able to define indefinite, possibly discontinuous, kernels, not necessarily inner product ones, that generate highly nonlinear separating surfaces.

Maximizing the distance between the separating planes in the higher dimensional space is surrogated in the present approach by support vector suppression, which is achieved by minimizing some desired norm of their Lagrange multipliers. The norm may be one induced by the separation kernel if it happens to be positive definite, or a Euclidean or a polyhedral norm (i.e., a norm induced by a bounded polyhedron such as the 1-norm or the ∞-norm). Polyhedral norms lead to linear programs whereas Euclidean norms lead to convex quadratic programs, all with an arbitrary separation kernel.

A standard support vector machine can be recovered by using the same kernel for separation and support vector suppression. On a simple test example, all models perform equally well when a positive definite kernel is used. When a negative definite kernel is used, we are unable to solve the nonconvex quadratic program associated with a conventional support vector machine, while all other proposed models remain convex and easily generate a surface that separates all given points.

8.1 Introduction

Support vector machines [Vapnik, 1995, Bennett and Blue, 1997, Girosi, 1998, Wahba, 1999b, Cherkassky and Mulier, 1998, Schölkopf, 1997, Smola, 1998] attempt to separate points belonging to two given sets in N-dimensional real (Euclidean) space $\mathbb{R}^N$ by a nonlinear surface, often only implicitly defined by a kernel function. In our approach here the nonlinear surface in the original input space $\mathbb{R}^N$ which is defined linearly in its parameters, can be represented as a linear function (plane) in a higher, often much higher dimensional feature space, say $\mathbb{R}^\ell$. Also, the original points of the two given sets can also be mapped into this higher dimensional space. If the two sets are linearly separable in $\mathbb{R}^\ell$, then it is intuitively plausible to generate a plane mid-way between the furthest parallel planes apart that bound the two sets. Using a distance induced by the kernel generating the nonlinear surface in $\mathbb{R}^N$, it can be shown [Vapnik and Lerner, 1963] that such a plane optimizes the generalization ability of the separating plane. If the two sets are not linearly separable, a similar approach can be used [Cortes and Vapnik, 1995, Vapnik, 1995] to maximize the distance between planes that bound each set with certain minimal error. Linear separation by planes with maximal 1-norm distance and by nonlinear surfaces were proposed earlier in [Mangasarian, 1965, 1968] as well as with soft margins [Bennett and Mangasarian, 1992]

In this paper we start with a nonlinear separating surface (8.1), defined by some arbitrary kernel k and by some linear parameters $u \in \mathbb{R}^m$, to be determined, that turn out to be closely related to some dual variables. Based on this surface we derive a general convex mathematical program (8.5) that attempts separation via the nonlinear surface (8.1) while minimizing some function θ of the parameters u. In our formulation here, the function θ which attempts to suppress u can be interpreted as minimizing the number of support vectors, or under more conventional assumptions as maximizing the distance between the separating planes in $\mathbb{R}^\ell$. The choice of θ leads to various support vector machines. We consider two classes of such machines based on whether θ is quadratic or piecewise linear. If we choose θ to be a quadratic function generated by the kernel defining the nonlinear surface (8.1), then we are led to the conventional dual quadratic program (8.9) associated with a support vector machine which requires positive definiteness of this kernel. However the quadratic function choice for θ can be divorced from the kernel defining the separating surface and this leads to other convex quadratic programs such as (8.10) *without* making any assumptions on the kernel. In [Smola and Schölkopf, 1998b] techniques for dealing with kernels that are not positive semidefinite were also presented. Another class of support vector machines are generated by choosing a piecewise linear convex function for θ and this leads to linear programs such as (8.11) and (8.12), both of which make no assumptions on the kernel. In Section 8.5 we give some simple applications of all four formulations to the Exclusive-Or (XOR) problem using first a positive definite second-order polynomial kernel and then a negative definite third-order polynomial kernel. For the positive definite kernel all four convex formulations

are easily solved and the resulting nonlinear surfaces separate all points in all cases. However, for the negative definite kernel, a powerful state-of-the-art package fails to solve the nonconvex quadratic program associated with the conventional support vector machine, whereas all other three convex formulations are easily solved and lead to complete separation of the data by the nonlinear surface.

A word about our notation and background material. All vectors will be column vectors unless transposed to a row vector by a prime superscript $'$. For a vector s in the N-dimensional real space $\mathbb{R}^N$, the step function s_* of $s \in \mathbb{R}^N$ is defined as a vector of ones and zeros in $\mathbb{R}^N$, with ones corresponding to positive components of s and zeros corresponding to nonpositive components. The scalar (inner) product of two vectors s and y in the N-dimensional real space $\mathbb{R}^N$ will be denoted by $s \cdot y$. For an $m \times N$ matrix X, X_i will denote the ith row of X and $X_{\cdot j}$ will denote the jth column of X. The identity matrix in a real space of arbitrary dimension will be denoted by I, while a column vector of ones of arbitrary dimension will be denoted by e. We shall employ the MATLAB [1992] "dot" notation to signify application of a function to all components of a matrix or a vector. For example if $X \in \mathbb{R}^{m \times N}$, then $X_\bullet^2 \in \mathbb{R}^{m \times N}$ will denote the matrix of elements of X squared.

We begin by defining a general kernel function as follows.

general kernels with no assumptions

Definition 8.1
Let $X \in \mathbb{R}^{m \times N}$ and $B \in \mathbb{R}^{N \times \ell}$. The **kernel** $k(X, B)$ maps $\mathbb{R}^{m \times N} \times \mathbb{R}^{N \times \ell}$ into $\mathbb{R}^{m \times \ell}$.

In particular if s and t are column vectors in $\mathbb{R}^N$ then, $k(s', X')$ is a row vector in $\mathbb{R}^m$, $k(s', t)$ is a real number and $k(X, X')$ is an $m \times m$ matrix. Note that for our purposes here $k(X, X')$ need not be symmetric in general. Examples of kernels are given in the introduction to the book and below where $a \in \mathbb{R}^m$, $b \in \mathbb{R}^\ell$, $\mu \in R$ and d is an integer. For simplicity we restrict ourselves here to finite dimensional kernels although many of the results can be extended to infinite dimensional Hilbert spaces [Kimeldorf and Wahba, 1971].

Example 8.1
Polynomial Kernel $(XB + \mu ab')_\bullet^d$

Example 8.2
Neural Network Step Kernel $(XB + \mu ab')_{\bullet *}$

Example 8.3
Radial Basis Kernel $e^{-\mu \|X_i' - B_{\cdot j}\|^2}$, $i, j = 1, \ldots, m$, $\ell = m$,
where, here only, e is the base of the natural logarithm.

Note that our approach allows discontinuous kernels such as the neural network step kernel with a discontinuous step function without the need for a smoothing approximation such as the sigmoid or hyperbolic tangent approximation as is usually done [Vapnik, 1995, Cherkassky and Mulier, 1998].

8.2 GSVM: The General Support Vector Machine

We consider a given set $\mathcal{A}$ of m points in real N-dimensional space of features $\mathbb{R}^N$ represented by the matrix $X \in \mathbb{R}^{m\times N}$. Each point X_i, $i = 1, \ldots, m$, belongs to class 1 or class -1 depending on whether Y_{ii} is 1 or -1, where $Y \in \mathbb{R}^{m\times m}$ is a given diagonal matrix of plus or minus ones. We shall attempt to discriminate between the classes 1 and -1 by a nonlinear *separating surface*, induced by some kernel $k(X, X')$, as follows:

GSVM separating surface

$$k(s', X')Y \cdot u = b, \tag{8.1}$$

where $k(s', X') \in \mathbb{R}^m$, according to Definition 8.1. The parameters $u \in \mathbb{R}^m$ and $b \in R$ are determined by solving a mathematical program, typically quadratic or linear. A point $s \in \mathbb{R}^N$ is classified in class 1 or -1 according to whether the *decision function*

$$(k(s', X')Y \cdot u - b)_*, \tag{8.2}$$

yields 1 or 0 respectively. The kernel function $k(s', X')$ defines a nonlinear map from $s \in \mathbb{R}^N$ to some other space $s \in \mathbb{R}^\ell$ where ℓ may be much larger than N. In particular if the kernel k is an inner product kernel under Mercer's condition [Courant and Hilbert, 1953, pp 138-140],[Vapnik, 1995, Cherkassky and Mulier, 1998, Burges, 1998] (an assumption that we will not make in this paper) then for s and t in $\mathbb{R}^N$:

$$k(s, t) = \Phi(s) \cdot \Phi(t), \tag{8.3}$$

and the separating surface (8.1) becomes:

$$\Phi(s)'\Phi(X')Y \cdot u = b, \tag{8.4}$$

where Φ is a function, not easily computable, from $\mathbb{R}^N$ to $\mathbb{R}^\ell$, and $\Phi(X') \in \mathbb{R}^{\ell\times m}$ results from applying Φ to the m columns of X'. The difficulty in computing Φ and the possible high dimensionality of $\mathbb{R}^\ell$ have been important factors in using a kernel k as a generator of a nonlinear separating surface in the original feature space $\mathbb{R}^N$ but which is linear in the high dimensional space $\mathbb{R}^\ell$. Our separating surface (8.1) written in terms of a kernel function retains this advantage and is linear in its parameters, u, b. We now state a mathematical program that generates such a surface for a general kernel k as follows:

the GSVM

$$\begin{array}{lrcl} \min\limits_{u,b,\xi} & Ce \cdot \xi + \theta(u) & & \\ \text{s.t.} & Y(k(X, X')Yu - eb) + \xi & \geq & e \\ & \xi & \geq & 0. \end{array} \tag{8.5}$$

Here θ is some convex function on $\mathbb{R}^m$, typically some norm or seminorm, and C is some positive parameter that weights the separation error $e \cdot \xi$ versus suppression of the separating surface parameter u. Suppression of u, utilized in [Bradley and

Mangasarian, 1998] for feature selection, can be interpreted in one of two ways. We interpret it here as minimizing the number of support vectors, i.e., constraints of (8.5) with positive multipliers. A more conventional interpretation is that of maximizing some measure of the distance or margin between the bounding parallel planes in $\mathbb{R}^\ell$, under appropriate assumptions, such as θ being a quadratic function induced by a positive definite kernel k as in (8.9) below. As is well known, this leads to improved generalization by minimizing an upper bound on the VC dimension [Vapnik, 1995, Schölkopf, 1997]. Girosi et al. [1993] used a quadratic function for the regularization term θ while Smola [1998] used linear and quadratic terms.

We term a solution of the mathematical program (8.5) and the resulting separating surface (8.1) and corresponding decision function (8.2) a *generalized support vector machine*, GSVM. In the following sections of the paper we derive a number of special cases, including the standard support vector machine. First, however, it is important to state under what conditions does the mathematical program (8.5) have a solution.

Proposition 8.2
Existence of a GSVM For any given $X \in \mathbb{R}^{m\times N}$, any $Y \in \mathbb{R}^{m\times m}$, $C > 0$ and any kernel k, the mathematical program (8.5) has a solution whenever θ is a piecewise-linear or quadratic function bounded below on $\mathbb{R}^m$.

Proof The feasible region of (8.5) is always nonempty. Just take: $u = 0$, $b = 0$ and $\xi = e$. When θ is piecewise-linear, existence follows from the standard linear programming result, that a feasible linear program with a bounded objective has a solution. Just apply this result to each piece of the objective on its polyhedral region. For a quadratic θ the result is a direct consequence of the Frank-Wolfe existence result for quadratic programming [Frank and Wolfe, 1956]. ■

We note that no convexity of θ was needed for this existence result. However in our specific applications where duality theory will be invoked, θ will need to be convex.

8.3 Quadratic Programming Support Vector Machines

We consider in this section support vector machines that include the standard ones [Vapnik, 1995, Cherkassky and Mulier, 1998, Burges, 1998] and which are obtained by setting θ of (8.5) to be a convex quadratic function $\theta(u) = \frac{1}{2}u \cdot Hu$, where $H \in \mathbb{R}^{m\times m}$ is some symmetric positive definite matrix. The mathematical program (8.5) becomes the following convex quadratic program:

$$\begin{array}{lrcl} \min\limits_{u,b,\xi} & Ce \cdot \xi + \frac{1}{2}u \cdot Hu & & \\ \text{s.t.} & Y(k(X, X')Yu - eb) + \xi & \geq & e \\ & \xi & \geq & 0. \end{array} \tag{8.6}$$

dual QP GSVM

The Wolfe [1961] dual [Mangasarian, 1994] of this convex quadratic program is:

$$\begin{array}{lrcl} \min\limits_{\boldsymbol{\alpha}\in\mathbb{R}^m} & \frac{1}{2}\boldsymbol{\alpha}\cdot Yk(X,X')YH^{-1}Yk(X,X')'Y\boldsymbol{\alpha} - e\cdot\boldsymbol{\alpha} & & \\ \text{s.t.} & e\cdot Y\boldsymbol{\alpha} & = & 0 \\ & 0\le\boldsymbol{\alpha} & \le & Ce. \end{array} \tag{8.7}$$

Furthermore, the primal variable u is related to the dual variable $\boldsymbol{\alpha}$ by:

$$u = H^{-1}Yk(X,X')'Y\boldsymbol{\alpha}. \tag{8.8}$$

Equations (8.6)-(8.8) have also been given by Smola and Schölkopf [1998b]. If we assume that the kernel $k(X,X')$ is symmetric positive definite and let $H = Yk(X,X')Y$, then our problem (8.6) becomes the standard SVM problem and our dual problem (8.7) degenerates to the dual problem of the standard support vector machine [Vapnik, 1995, Cherkassky and Mulier, 1998, Burges, 1998] with $u = \boldsymbol{\alpha}$:

$$\begin{array}{lrcl} \min\limits_{\boldsymbol{\alpha}\in\mathbb{R}^m} & \frac{1}{2}\boldsymbol{\alpha}\cdot Yk(X,X')Y\boldsymbol{\alpha} - e\cdot\boldsymbol{\alpha} & & \\ \text{s.t.} & e\cdot Y\boldsymbol{\alpha} & = & 0 \\ & 0\le\boldsymbol{\alpha} & \le & Ce. \end{array} \tag{8.9}$$

dual SVM

The positive definiteness assumption on $k(X,X')$ in (8.9) can be relaxed to positive *semi*definiteness while maintaining the convex quadratic program (8.6), with $H = Yk(X,X')Y$, as the direct dual of (8.9) without utilizing (8.7) and (8.8). The symmetry and positive semidefiniteness of the kernel $k(X,X')$ for this version of a support vector machine is consistent with the support vector machine literature. The fact that $\boldsymbol{\alpha} = u$ in the dual formulation (8.9), shows that the variable u appearing in the original formulation (8.6) is also the dual multiplier vector for the first set of constraints of (8.6). Hence the quadratic term in the objective function of (8.6) can be thought of as suppressing as many multipliers of support vectors as possible and thus minimizing the number of such support vectors. This is another interpretation of the standard support vector machine that is usually interpreted as maximizing the margin or distance between parallel separating planes.

This leads to the idea of using other values for the matrix H other than $Yk(X,X')Y$ that will also suppress u. One particular choice is interesting because it puts no restrictions on k: no symmetry, no positive definiteness or semidefiniteness and not even continuity. This is the choice $H = I$ in (8.6) which leads to a dual problem (8.7) with $H = I$ and $u = Yk(X,X')'Y\boldsymbol{\alpha}$ as follows:

arbitrary-kernel dual QP GSVM

$$\begin{array}{lrcl} \min\limits_{\boldsymbol{\alpha}\in\mathbb{R}^m} & \frac{1}{2}\boldsymbol{\alpha}\cdot Yk(X,X')k(X,X')'Y\boldsymbol{\alpha} - e'\boldsymbol{\alpha} & & \\ \text{s.t.} & e\cdot Y\boldsymbol{\alpha} & = & 0 \\ & 0\le\boldsymbol{\alpha} & \le & Ce. \end{array} \tag{8.10}$$

Note that setting $H = I$ corresponds to weight decay in neural networks and as shrinkage estimators in statistics [Smola and Schölkopf, 1998b]. We also note that $k(X,X')k(X,X')'$ is positive semidefinite with no assumptions on $k(X,X')$, and hence the above problem is an always solvable convex quadratic program

for any kernel $k(X, X')$. In fact by Proposition 8.2 the quadratic program (8.6) is solvable for *any* symmetric positive semidefinite matrix H, and by quadratic programming duality so is its dual problem (8.7), the solution $\boldsymbol{\alpha}$ of which and $u = Yk(X, X')'Y\boldsymbol{\alpha}$ can be immediately used to generate a decision function (8.2). Thus we are free to choose any symmetric positive definite matrix H to generate a support vector machine. Experimentation will be needed to determine what are the most appropriate choices for H.

Note that even though (8.10) differs from (8.9) merely in the replacement of the kernel $k(X, X')$ by $k(X, X')k(X, X')'$, the two problems lead to distinct separating surfaces (8.1) because $u = \boldsymbol{\alpha}$ for (8.9) and $u = Yk(X, X')'Y\boldsymbol{\alpha}$ for (8.10). Note that some sparsity in the solution $\boldsymbol{\alpha}$ of (8.10) may be lost with the product kernel if it is not positive definite.

We turn our attention to linear programming support vector machines.

8.4 Linear Programming Support Vector Machines

In this section we consider problems generated from the mathematical program (8.5) by using a piecewise linear function θ in the objective function thus leading to linear programs.

LP1 GSVM with arbitrary kernel

The most obvious choice for θ is the 1-norm of u, which leads to the following linear programming formulation:

$$\begin{array}{rrcl} \min\limits_{u,b,\xi,t} & Ce \cdot \xi + e \cdot t & & \\ \text{s.t.} & Y(k(X, X')Yu - eb) + \xi & \geq & e \\ & t \geq u & \geq & -t \\ & \xi & \geq & 0. \end{array} \tag{8.11}$$

A solution (u, b, ξ, t) to this linear program for a chosen kernel $k(X, X')$ will provide a decision function as given by (8.2). This linear program parallels the quadratic programming formulation (8.10) that was obtained as the dual of (8.5) by setting $\theta(u)$ therein to half the 2-norm squared of u whereas $\theta(u)$ is set to the 1-norm of u in (8.11). Another linear programming formulation that somewhat parallels the quadratic programming formulation (8.9), which was obtained as the dual of (8.5) by setting $\theta(u)$ therein to half the 2-norm squared of $k(X, X')^{\frac{1}{2}}Yu$, is obtained setting θ to be the 1-norm of $k(X, X')Yu$. The motivation for this idea is to try capturing a norm induced by $k(X, X')$ even when the kernel is not positive semidefinite. This leads to the following linear program:

LP2 GSVM with arbitrary kernel

$$\begin{array}{rrcl} \min\limits_{u,b,\xi,t} & Ce \cdot \xi + e \cdot t & & \\ \text{s.t.} & Y(k(X, X')Yu - eb) + \xi & \geq & e \\ & t \geq k(X, X')Yu & \geq & -t \\ & \xi & \geq & 0. \end{array} \tag{8.12}$$

No assumptions of symmetry or positive definiteness on $k(X, X')$ are needed in either of the above linear programming formulations as was the case in the quadratic program (8.9).

It is interesting to note that if the linear kernel $k(X, X') = XX'$ is used in the linear program (8.11) we obtain the high-performing 1-norm linear SVM proposed by Bredensteiner and Bennett [1997] and utilized successfully in [Bredensteiner, 1997, Bennett et al., 1998, Bradley and Mangasarian, 1998]. Hence, if we set $w = X'Yu$ in (8.11) we obtain [Bradley and Mangasarian, 1998, Equation (13)].

8.5 A Simple Illustrative Example

We first demonstrate the workings and sometimes different, yet equally effective, decision surfaces obtained by the various proposed mathematical programming formulations, for a positive definite symmetric kernel. We then show that for a negative definite symmetric kernel, the conventional support vector machine fails to generate a decision function that correctly separates the given points, whereas all the new formulations do succeed in generating a decision surface that correctly separates all the given points.

For our positive definite kernel we use a polynomial kernel of order 2, based on Example 8.1 with $B = X'$, $\mu = 1$, $a = b = e$ and $d = 2$, and apply it to the classical Exclusive-Or (XOR) problem. We thus have:

$$X = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ -1 & -1 \\ -1 & 1 \end{bmatrix}, \; Y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}. \tag{8.13}$$

Hence with the MATLAB [1992] "dot" notation signifying componentwise exponentiation we get that:

$$k(X, X') = (XX' + ee')^2_\bullet = \begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix}, \tag{8.14}$$

and

$$k(s', X') = (s'X' + e')^2_\bullet = [s_1 + s_2 + 1 \;\; s_1 - s_2 + 1 \;\; -s_1 - s_2 + 1 \;\; -s_1 + s_2 + 1]^2_\bullet. \tag{8.15}$$

Solution of the linear program (8.11) with $C \geq 1$ gives:

$$u = \frac{1}{8}e, \; b = 0, \; \xi = 0, \; t = \frac{1}{8}e, \; Ce \cdot \xi + e \cdot t = \frac{1}{2}. \tag{8.16}$$

Note that the $\xi = 0$ in the above solution means that the decision surface correctly classifies all given points represented by X and Y. Solution of either quadratic program (8.9) or (8.10) with the same kernel and for $C \geq 1$ also yields $u = \frac{1}{8}e$. Substitution of this u in (8.6) and solving the resulting linear program gives the same b, ξ as in (8.16). Thus all mathematical programs (8.9), (8.10) and (8.11) yield exactly the same decision surface (8.2):

$$(k(s', X')Yu - b)_* = ((s'X' + e')^2_\bullet Yu - b)_* = (s_1 s_2)_*, \tag{8.17}$$

a step function of the quadratic function $s_1 s_2$, which correctly classifies the two categories class 1 and class -1 and is in agreement with the solution obtained in [Cherkassky and Mulier, 1998, pages 372-375] for the conventional support vector machine (8.9). Note that neither mathematical program (8.10) or (8.11) required positive definiteness of $k(X, X')$, whereas (8.9) does.

However, it is rather interesting to observe that the linear programming solution (8.16) is not unique. In fact another solution is the following:

$$u = \begin{bmatrix} 0 \\ \frac{1}{4} \\ 0 \\ \frac{1}{4} \end{bmatrix}, \; b = -\frac{3}{2}, \; \xi = 0, \; t = \begin{bmatrix} 0 \\ \frac{1}{4} \\ 0 \\ \frac{1}{4} \end{bmatrix}, \; Ce \cdot \xi + e \cdot t = \frac{1}{2}. \tag{8.18}$$

For this solution the decision surface (8.2) turns out to be:

$$(k(z', X')Yu - b)_* = ((s'X' + e')^2_\bullet Yu - b)_* = \frac{1}{2}(2 - (s_1 - s_2)^2)_*. \tag{8.19}$$

This decision surface is rather different from that of (8.17), but it does separate the two classes correctly and in fact it consists of two parallel lines separating $\mathbb{R}^2$ into 3 regions, whereas (8.17) separates $\mathbb{R}^2$ into four quadrants each pair of which contains one class. Both of these decision functions are depicted in Figure 8.1 and Figure 8.2.

Solution of the linear program (8.12) with $C > 1$ yields:

$$u = \begin{bmatrix} \frac{1}{24} \\ \frac{5}{24} \\ \frac{1}{24} \\ \frac{5}{24} \end{bmatrix}, \; b = -1, \; \xi = 0, \; t = \begin{bmatrix} 0 \\ 2 \\ 0 \\ 2 \end{bmatrix}, \tag{8.20}$$

which gives the decision surface:

$$((s'X' + e')^2_\bullet Yu - b)_* = \frac{1}{3}(2 + s_1 s_2 - (s_1 - s_2)^2)_*. \tag{8.21}$$

This decision function divides $\mathbb{R}^2$ into three regions by a pair of "square root" curves that correctly classify the two classes as depicted in Figure 8.3.

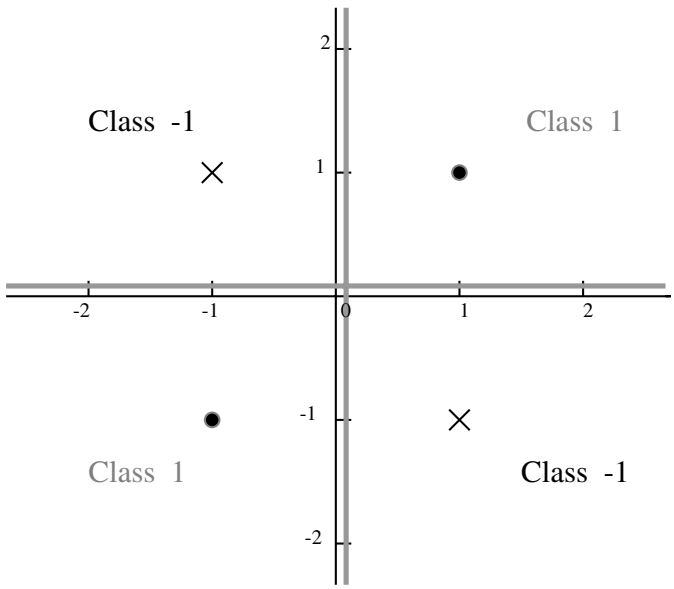


Figure 8.1 XOR discrimination by a step function of a quadratic function: $(s_1 s_2)_*$ obtained by the linear program (8.11) and the quadratic programs (8.9) and (8.10).

Finally in order to show that positive definiteness of the kernel $k(X, X')$ is not essential in any of our new mathematical programming formulations (8.10), (8.11) or (8.12), and whereas it is in the conventional quadratic programming formulation (8.9), we consider the following negative definite kernel:

$$k(X, X') = (-XX' - ee')^3_\bullet, \tag{8.22}$$

and attempt to solve the mathematical programs (8.9),(8.10), (8.11) and (8.12) with this kernel and with $C = 1$. The powerful PATH mathematical programming package [Ferris and Munson, 1999, Dirkse and Ferris, 1995] failed to solve the nonconvex quadratic programming formulation (8.9) for the conventional support vector machine. (PATH is a Newton-based method for solving a nonsmooth systems of equations that subsume the Karush Kuhn Tucker conditions of quadratic and nonlinear programming.) In contrast, the same package solved the quadratic program (8.10) giving $\boldsymbol{\alpha} = \frac{1}{576}e$ and a corresponding $u = Yk(X, X')'Y\boldsymbol{\alpha} = -\frac{1}{24}e$. Substitution of this u in the quadratic program (8.6) and solving the resulting linear program gives: $b = 0,\ \xi = 0$. The solution $\xi = 0$ indicates that all points represented by X have been correctly classified, which is corroborated by the resulting decision surface $(s_1 s_2)_*$, the same as that of (8.17). This indicates the effectiveness of the quadratic program (8.10) in its ability to extract from the negative definite cubic kernel just the required quadratic term to achieve correct separation. Similarly both linear programs (8.11) and (8.12) gave $\xi = 0$ thus also achieving complete separation with this negative definite kernel.

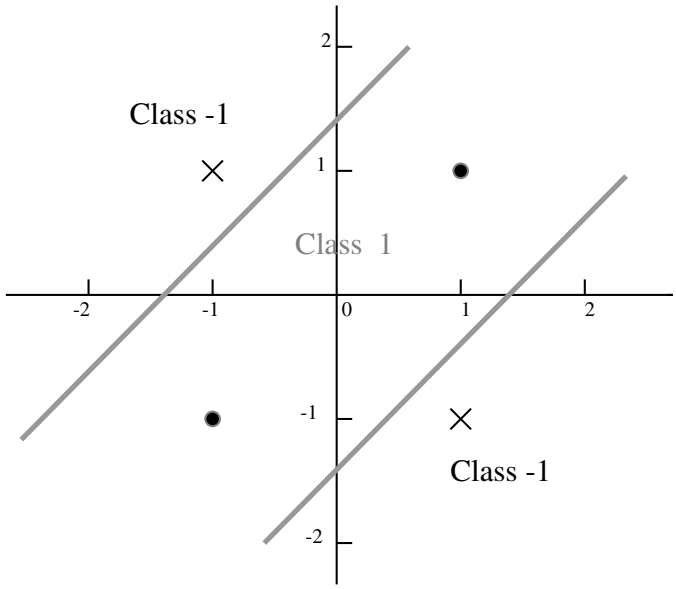


Figure 8.2 XOR discrimination by a step function of a quadratic function: $(2 - (s_1 - s_2)^2)_*$ obtained by another solution of the linear program (8.11).

8.6 Conclusion

We have proposed a direct mathematical programming framework for general support vector machines that makes essentially no or few assumptions on the kernel employed. We have derived new kernel-based linear programming formulations (8.11) and (8.12), and a new quadratic programming formulation (8.10) that require no assumptions on the kernel k. These formulations can lead to effective but different decision functions from those obtained by the quadratic programming formulation (8.9) for a conventional support vector machine that requires symmetry and positive definiteness of the kernel. Even for negative definite kernels these new formulations can generate decision functions that separate the given points whereas the conventional support vector machine does not. This leads us to suggest that further testing and experimentation with mathematical programming formulations such as (8.11), (8.12) and (8.10) and others are worthwhile. These formulations may open the way for a variety of support vector machines that could be tested computationally against each other and against existing ones. Furthermore, broad classes of serial and parallel optimization algorithms such as [Bennett and Mangasarian, 1994] can be brought to bear on these different formulations exploiting their structure in order to permit the processing of massive databases.

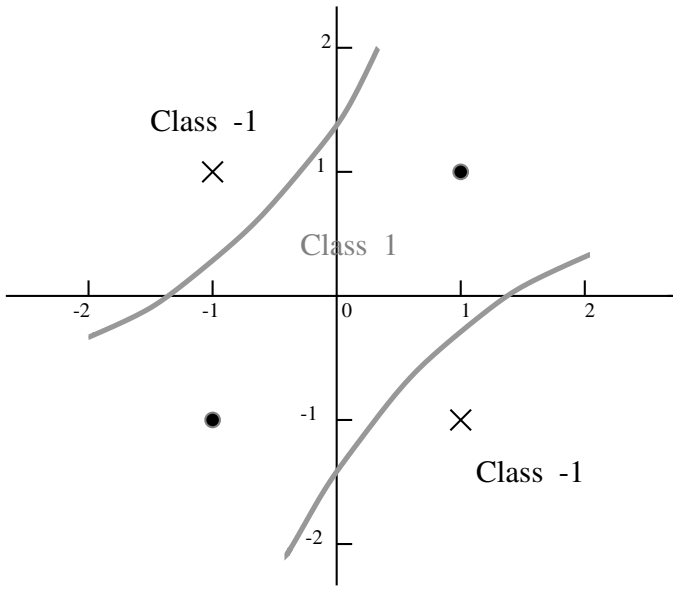


Figure 8.3 XOR discrimination by a step function of a quadratic function: $(2 + s_1 s_2 - (s_1 - s_2)^2)_*$ obtained by the linear program (8.12).

Acknowledgments

I am grateful to my colleague Grace Wahba for very helpful discussions on kernel theory, to my PhD student David Musicant and to Paul Bradley for important comments on the paper.

This research is supported by National Science Foundation Grants CCR-9322479, CCR-9729842 and CDA-9623632, and by Air Force Office of Scientific Research Grant F49620-97-1-0326 as Mathematical Programming Technical Report 98-14, October 1998.

9 Linear Discriminant and Support Vector Classifiers

Isabelle Guyon
Clopinet
955 Creston Road
Berkeley, CA 94708-1501
isabelle@clopinet.com
http://www.clopinet.com

David G. Stork
Ricoh Silicon Valley
2882 Sand Hill Road, Suite 115
Menlo Park, CA 94025-7022
stork@rsv.ricoh.com
http://rsv.ricoh.com/~ stork

Support Vector Machines were introduced and first applied to classification problems as alternatives to multi-layer neural networks. The high generalization ability provided by Support Vector Classifiers (SVCs) has inspired recent work on computational speedups as well as the fundamental theory of model complexity and generalization. At first glance, a Support Vector Classifier appears to be nothing more than a generalized linear discriminant in a high dimensional transformed feature space; indeed, many aspects of SVCs can best be understood in relation to traditional linear discriminant techniques.

This chapter explores interconnections between many linear discriminant techniques, including Perceptron, Radial Basis Functions (RBFs) and SVCs. The principle of duality between learning- or feature-based techniques (such as Perceptrons) and memory- or example-based methods (such as RBFs) is central to the development of SVCs. We provide several other examples of duality in linear discriminant learning algorithms.

9.1 Introduction

Support Vector Classifiers are linear classifiers in a high dimensional space. Recently, a large number of SVC training algorithms have been proposed [Krauth and Mézard, 1987, Frieß et al., 1998, Freund and Schapire, 1998, Schölkopf et al., 1999a], but there seems to be a lack of appreciation of the close relationships among SVCs and many other classification techniques, in particular older "classical" methods. Our goal is to clarify key relationships among several important methods. While we cannot address all relevant techniques, we shall discuss methods that are of historical importance, practical use, and illustrative of key concepts. In doing so, we shall emphasize the complementary nature of distributed *learning based methods* in a direct weight space, and local, *memory based methods* in a dual parameter space. We shall not consider questions of regularization, generalization, structural risk minimization, or convergence rates.

There are three interrelated reasons for the long and extensive literature on linear classifiers. First, such classifiers are computationally quite simple, both during learning and during classification. Second, despite this simplicity, linear classifiers have had great success in a large number of application areas. Finally, they are amenable to theoretical analysis. For instance, the proof of the Perceptron convergence theorem greatly encouraged practitioners [Rosenblatt, 1958]. These reasons continue to underlay the interest in current linear methods such as SVCs.

Our chapter is organized as follows. In Section 9.2 we review the basic concepts of classifiers based on linear discriminants and highlight the key role of duality between learning methods in the direct space and those in a dual space. We turn, in Section 9.3, to several settings or formulations of the learning problem for classification, including those based on criteria of training error, Perceptron objective function and mean squared error objective function. Then in Section 9.4 we describe several training algorithms for linear classifiers in general and SVCs in particular. Then in Section 9.5 we consider the properties of the solutions, such as robustness, sensitivity to outliers, and so on. We conclude in Section 9.6 with a summary of our main points and a set of outstanding problems.

9.2 What is a Linear Discriminant?

9.2.1 Classification Problem

We consider the two class classification problem, in which patterns, represented as n-dimensional vectors $\mathbf{x}$, are labeled with binary values $y \in \{-1, +1\}$. For example, assume that images of handwritten digits `0` and `1` must be classified. An optical scanner yields a gray level pixel image; each pixel can be encoded as a number corresponding to one component of $\mathbf{x}$. Images of the digit `0` are labeled -1 and images of the digit `1` are labeled $+1$.

Classification problems are usually characterized by an unknown probability distribution $p(\mathbf{x}, y)$ on $\mathbb{R}^n \times \{-1, +1\}$, but training examples $\mathbf{x}_i$ and their corresponding class labels y_i are provided:

$$X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subseteq \mathbb{R}^n \tag{9.1}$$

$$Y = \{y_1, \ldots, y_m\} \subseteq \{-1, 1\}. \tag{9.2}$$

The problem is to find a decision function $g(\mathbf{x})$, which predicts accurately the class label y of any example $\mathbf{x}$ that may or may not belong to the training set.

9.2.2 Discriminant Function

discriminant function

The discriminant function approach uses a real valued function $f(\mathbf{x})$, called *discriminant function*, the sign of which determines the class label prediction: $g(\mathbf{x}) = \operatorname{sgn}(f(\mathbf{x}))$. The discriminant function $f(\mathbf{x})$ may be parametrized with some parameters $\mathbf{a} = \{a_1, \ldots, a_p\}$ that are determined from the training examples by means of a learning algorithm. To be explicit, we should write $f(\mathbf{x}; \mathbf{a})$, but we shall generally omit this in the following. A great many pattern classification methods employ discriminant functions, either implicitly or explicitly. For instance, when an input pattern $\mathbf{x}$ is presented to the input layer in a standard three layer neural network having a 1-of-c representation at the output layer, each output unit computes, effectively, a discriminant function. The pattern is assigned the category label corresponding to the output unit with the largest discriminant value.

9.2.3 Linear Discriminant

Linear discriminant functions are discriminant functions that are linear in their parameters. Two kinds of linear discriminant functions have been studied in depth:

$$f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}) \qquad \text{linear classifier or "Perceptron," and} \tag{9.3}$$

$$f(\mathbf{x}) = \sum_{i=1}^{m} y_i \alpha_i k(\mathbf{x}, \mathbf{x}_i) \quad \text{kernel classifier,} \tag{9.4}$$

with parameters $\mathbf{w} \in \mathbb{R}^N$, $\alpha_i \in \mathbb{R}$, basis functions $\Phi : \mathbb{R}^n \to \mathbb{R}^N$ and symmetric kernel functions $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$.[1] The basis and kernel functions may also be parametrized, but the parameters are not subject to training with the examples in the framework that we consider here.

For certain sets of basis functions, linear discriminant functions are universal approximators, that is, for well chosen parameters, the decision boundary $f(\mathbf{x}) = 0$

1. We simplify the parametrization of linear classifiers by integrating an eventual bias value b as an additional component of the weight vector. This corresponds to assuming that one of the components of $\Phi(\mathbf{x})$ is a constant.

can arbitrarily well approximate any decision boundary, including the ideal decision boundary. Such is the case, for instance, of polynomial classifiers. The basis functions $\Phi_k(\mathbf{x})$ of a polynomial classifier are monomials that are products of a number of components of $\mathbf{x}$.

9.2.4 Feature-based Classifiers

Perceptron

The basis functions $\Phi_k(\mathbf{x})$ of linear classifiers or "Perceptrons" of Equation 9.4 can be understood as feature detectors. For example, in a handwritten digit recognition task where 0s and 1s have to be classified, algorithms can be derived that detect the presence or absence of a straight line or of a loop. In the absence of domain knowledge about the task necessary to design such *ad hoc* feature detectors, general purpose basis functions can be used, such as monomials in a polynomial classifier.

9.2.5 Example-based Classifiers

RBF

The symmetric kernel functions $k(\mathbf{x}, \mathbf{x}')$ of kernel classifiers of Equation 9.4 are often (although not necessarily) radial basis functions. For example $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$ is a bell shaped or "Gaussian" kernel. Each contribution to the decision function $k(\mathbf{x}, \mathbf{x}_i)$ is centered around one example $\mathbf{x}_i$, where it is maximum, and vanishes as the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}_i$ increases. Several examples of general purpose kernel functions are given in Table 9.1. Note that the polynomial kernel is not, in general, a Radial Basis Function. The computation of the linear discriminant of kernel classifiers requires storing in memory the training examples, thus the name "example-based" or "memory-based" classifier.

Parzen windows

There are close relationships between kernel classifiers and other non-parametric classification methods such as Parzen windows [Parzen, 1962b] and nearest neighbors [Cover and Hart, 1967]. Parzen windows are essentially kernel classifiers normalized such that $f(\mathbf{x})$ can be used as an estimator of the posterior probability $P(y = 1|\mathbf{x})$. When rectangular windows are used, the classification decision is done according to the largest number of examples of each class falling within a window of a given size that is centered around $\mathbf{x}$.

nearest neighbors

Similarly, in a k nearest neighbor classifier, the decision is made according to the majority label among the k training examples that are nearest to the test point $\mathbf{x}$. One of the benefits of k nearest neighbor classifiers is that decision boundaries are smooth where the data is sparse, but sharp and curvy where the data is dense, as we might wish.

tangent distance

Kernels can also be thought of as similarity measures and thus can often be designed to incorporate domain knowledge about the task. For example, in a handwriting digit recognition task, a similarity measure that incorporates invariance with respect to rotations, translations and other distortions is better suited than a simple dot product (see for instance the "tangent distance" [Simard et al., 1993, Schölkopf et al., 1998a]).

9.2.6 Feature and Example Selection

In general, the more complex the classification problem (i.e., the more complex the optimal decision boundary), the larger the number of features that should be included in the classifier, for example all the monomials of a polynomial classifier of order d. However the number of parameters to estimate rapidly becomes prohibitive, e.g., $O(n^d)$. Similarly, training a large number of parameters may be infeasible, either because of high computation complexity or, more seriously, if there is insufficient training data, because the solution is underdetermined.

The classical approach to these general problems is to select a subset of features, itself often a subtle and challenging task [Schürmann, 1996]. One of the most popular methods is principal component analysis, in which the n-dimensional features are projected to a subspace so as to minimize an MSE measure [Oja, 1983]. Likewise, the computational load imposed by kernel classifiers can be reduced by selecting a subset of the training examples or by creating prototypes out of linear combinations of examples.

principal component analysis

9.2.7 Locality

Feature-based linear classifiers are considered "global" decision functions because they are globally defined by *all* the training examples. Conversely, kernel classifiers, which rely on local radial kernels, each of which determined by a local subset of the training examples, are often considered "local" classifiers. When radial kernels are used, the sign of the decision function $f(\mathbf{x})$ is influenced by many local soft decisions of presence or absence of $\mathbf{x}$ in the neighborhood of $\mathbf{x}_i$.

9.2.8 Duality

Under certain conditions linear classifiers, Equation 9.4, and kernel classifiers, Equation 9.4, are two different representations of the same discriminant function. This invalidates the usual distinction between "global" and "local" linear discriminant.

For example, a wide variety of training algorithms of linear classifiers, including gradient descent algorithms, yield a weight vector $\mathbf{w}$, which is a weighted sum of the training examples in Φ-space:

$$\mathbf{w} = \sum_{i=1}^{m} y_i \alpha_i \Phi(\mathbf{x}_i). \tag{9.5}$$

By substituting $\mathbf{w}$ in the expression of the linear classifier (Equation 9.4): $f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x})$ one obtains a kernel classifier:

$$f(\mathbf{x}) = \sum_{i=1}^{m} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) \tag{9.6}$$

with kernel $k(\mathbf{x}_i, \mathbf{x}) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})$.

$k(\mathbf{x}, \mathbf{x}')$	**Name**
$(\mathbf{x} \cdot \mathbf{x}' + 1)^d$	Polynomial of order d
$\exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$	Gaussian radial basis function
$\exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|)$	Exponential radial basis function

Table 9.1 Examples of kernels. Increasing parameters $d \in \mathbb{N}$ or $\gamma \in \mathbb{R}$ increases the capacity of the classifier.

Reciprocally, many kernels admit series expansions (eventually infinite), and can therefore be written as a dot product (see Mercer's conditions in Theorem 1.16).

Mercer conditions

Such is the case, for instance, of the polynomial and the Gaussian kernel (Table 9.1).

In the following, we will refer to the linear classifier (Perceptron) as the primal representation of the linear discriminant and to the corresponding kernel classifier as its dual. Therefore, $\mathbf{w}$ will be the primal parameters and $\alpha_i, 1 \leq i \leq m$ the dual parameters.

dual parameters

Depending on the number of features and the number of training examples, it may be beneficial to carry out the computations in one or the other representation. For instance, the polynomial kernel: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + 1)^d$ expands into the dot product between Φ vectors whose components contain all possible products of up to d components of $\mathbf{x}$ vectors. The number of primal parameters to estimate $N \simeq n^d$ (the dimension of feature space) is often much larger than the number of dual parameters m (the number of training examples).

9.2.9 Linear Separability

Besides being linear in its parameters, the linear classifier is linear in its input components in the N-dimensional Φ-space. Thus it defines a linear decision boundary (an $(N-1)$-dimensional hyperplane). A classification problem is said to be linearly separable, for given basis functions $\Phi(\mathbf{x})$, if there exists a parameter vector $\mathbf{w}$ such that $g(\mathbf{x}) = \operatorname{sgn}(\mathbf{w} \cdot \Phi(\mathbf{x}))$ classifies all vectors $\mathbf{x}$ without error.

Non-linear separability can arise from a number of causes (Figure 9.2.10):

- Classes may overlap in feature space (case (a)), due to the existence of inherently *ambiguous* patterns, e.g., a vertical bar can either represent a digit `1` or an lowercase `l`). Separability can also be due to the presence of "noisy data," including *meaningless* patterns and *mislabeled* valid examples. In the case of overlapping classes, the ideal decision boundary may nevertheless be linear. For instance, two classes generated by Gaussian distributions of identical covariance matrices have a Bayes optimum decision boundary $g(\mathbf{x}) = \operatorname{sgn}(p(\mathbf{x}, y = 1) - p(\mathbf{x}, y = -1))$ that is linear.
- Classes may have an ideal decision boundary which is not linear (case (b)), as might arise in the case of complex multimodal distributions.

From the point of view of finding an optimum linear discriminant function, these two cases are quite different. In case (a), one must allow that some examples will be

misclassified, i.e., that the asymptotic error rate of Bayes optimum decision function (Bayes error) is non zero. In case (b), the Bayes error is zero, and a better choice of representation may render the problem separable.

9.2.10 Linearly Separable Training Set

A linearly separable training set is not by itself an indication that the problem is linearly separable. Any sparse training set is likely to be linearly separable. A non linearly separable training set indicates that the problem is non linearly separable (provided that the training data does not contain any error). However, it does not provide a clue as to whether classes overlap or not.

Given a universal set of basis functions, it is always possible to separate the training set without error, for some value of the parameter of the associated kernel assuming no two identical training points have been assigned to different classes. For instance, one can always reduce the width (increase γ) of the Gaussian kernels centered on the training points. However, this approach tends to overfit the data and give poor classification error rate on unseen test data. Such overfitting can easily be understood since, in the limit of infinitely narrow kernels, any training set can be learned without error, but no prediction can be made on unseen data. The training examples are merely "memorized."

A general principle from learning theory predicts that for two classifiers having the same training error, the classifier with smaller capacity is more likely to perform better on unseen data [Vapnik, 1979]. Loosely speaking, the capacity is related to the number of training examples that can be separated without error. Therefore, increasing the kernel parameter allows to learn easily more training examples, but simultaneously increases capacity. Introducing some errors on the training set by reducing the capacity may yield a better classifier.

Note that this is completely independent on whether the data is "noisy" or not.

9.3 Formulations of the Linear Discriminant Training Problem

The goal of linear discriminant function training is to adjust its parameters such that the expected value of the classification error on unseen patterns (referred to as expected risk or prediction error) is minimized. Note that only the parameters $\mathbf{w}$ or $\boldsymbol{\alpha}$ are being trained in this framework. The additional parameters of the basis functions or kernel functions are not subject to training.

Under the assumption that the data is generated by a probability distribution $p(\mathbf{x}, y)$, the prediction error is given by:

$$R(f) = \int_{\mathbb{R}^N \times \{-1,1\}} 1_{\{g(\mathbf{x}) \neq y\}} \; p(\mathbf{x}, y) d\mathbf{x} dy. \tag{9.7}$$

where $g(\mathbf{x}) = \operatorname{sgn}(f(\mathbf{x}))$ and $f(\mathbf{x})$ is a linear discriminant function.

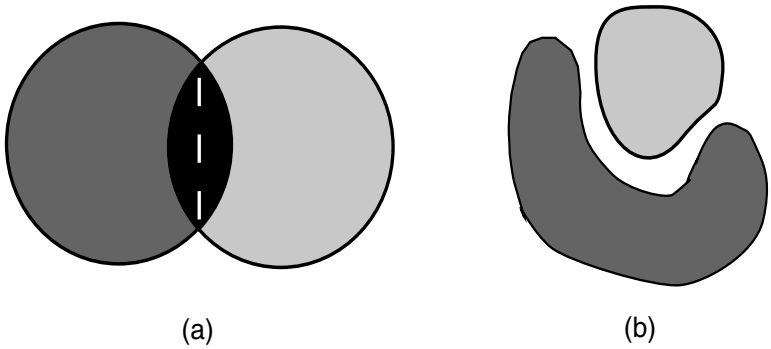


Figure 9.1 Non-linear separability. (a) Overlapping classes. The optimum decision boundary may still be linear. (b) Non overlapping classes. In the case shown, the optimum decision boundary is not linear.

The prediction $R(f)$ cannot be computed when $p(\mathbf{x}, y)$ is unknown, it can only be approximated. The various training methods that have been proposed all use a particular way of approximating $R(f)$, which is always optimum in some sense. We have selected a subset of methods that have some interesting connections to Support Vector Classifiers.

9.3.1 Minimizing the Number of Training Errors

The first method consists in estimating $R(f)$ by its discrete approximation computed with the training examples (the empirical risk or training error):

$$R_{emp}(f) = \sum_{i=1}^{m} 1_{\{g(\mathbf{x}_i) \neq y_i\}}. \tag{9.8}$$

The empirical risk functional $R_{emp}(f)$ is piecewise constant, which is typically difficult to optimize by standard techniques such as gradient descent.

9.3.1.1 Perceptron Objective Function

Perceptron objective function

One of the most popular substitute objective functions is the Perceptron objective function. The Perceptron objective function is equal to the sum of the "margin values" of the misclassified examples:

$$J_{\text{Perceptron}}(f) = -\sum_{i \in \mathcal{M}} y_i f(\mathbf{x}_i) = -\sum_{i \in \mathcal{M}} \rho_f(\mathbf{x}_i, y_i) \tag{9.9}$$

where $\mathcal{M} = \{i, i = 1, \ldots, m : g(\mathbf{x}_i) = \text{sgn}\,(f(\mathbf{x}_i)) \neq y_i\}$ is the set of misclassified examples and $\rho_f(\mathbf{x}_i, y_i) = y_i f(\mathbf{x}_i) = y_i \mathbf{w} \cdot \Phi(\mathbf{x})$ are the margin values. The margin values are proportional to the distances from the misclassified examples to the

decision boundary in Φ-space. The misclassified examples have negative margin values: $\rho_f(\mathbf{x}_i, y_i) < 0$.

If the training set is linearly separable, a set of parameters that minimizes $J_{\text{Perceptron}}(f)$ also minimizes $R_{emp}(f)$ of Equation 9.8. If the training set is not linearly separable, the minimization of $R_{emp}(f)$ is not guaranteed, but a set of parameters that minimizes $J_{\text{Perceptron}}(f)$ may nevertheless yield an acceptable practical solution.

9.3.2 Approximating the Bayes Optimum Discriminant Function

Bayes optimum discriminant

A different approach is to approximate the Bayes optimum discriminant function by a linear discriminant. By minimizing an approximation of $R(f)$, we were trying to find a decision function $g(\mathbf{x}) = \text{sgn}(f(\mathbf{x}))$, where $f(\mathbf{x})$ is a linear discriminant, that minimizes the number of classification errors. We now consider the decision function $g_{\text{Bayes}}(\mathbf{x}) = \text{sgn}(f_{\text{Bayes}}(\mathbf{x}))$, which gives the theoretical minimum number of classification errors. It is based on a discriminant function, e.g., $f_{\text{Bayes0}}(\mathbf{x}) = p(\mathbf{x}, y = 1) - p(\mathbf{x}, y = -1)$, which we approximate with a linear discriminant function. We have therefore replaced the problem of training a classifier by that of estimating a probability density.

Any discriminant function $f(\mathbf{x})$ can be transformed by a monotonically increasing function without changing the outcome of the decision function $g(\mathbf{x}) = \text{sgn}(f(\mathbf{x}))$. The Bayes decision function can therefore be constructed with several discriminant functions:

$$f_{\text{Bayes0}}(\mathbf{x}) = p(\mathbf{x}, y = 1) - p(\mathbf{x}, y = -1) \tag{9.10}$$

$$f_{\text{Bayes1}}(\mathbf{x}) = P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x}) \tag{9.11}$$

$$f_{\text{Bayes2}}(\mathbf{x}) = \log\left(\frac{p(\mathbf{x}|y = 1)}{p(\mathbf{x}|y = -1)}\right). \tag{9.12}$$

9.3.2.1 MSE Objective Function

A first method consists in seeking the linear discriminant $f(\mathbf{x})$ that best approximates $f_{\text{Bayes1}}(\mathbf{x}) = P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$ in the least squares sense. This function corresponds to the minimum of the objective function:

$$\int_{\mathbb{R}^N \times \{-1,1\}} (f_{\text{Bayes1}}(\mathbf{x}) - f(\mathbf{x}))^2 p(\mathbf{x}, y) d\mathbf{x} dy. \tag{9.13}$$

This problem seems to require the knowledge of $f_{\text{Bayes1}}(\mathbf{x})$. However, it can be shown (see, e.g., [Duda and Hart, 1973]) that the linear discriminant that minimizes this objective function also minimizes:

$$\int_{\mathbb{R}^N \times \{-1,1\}} (y - f(\mathbf{x}))^2 p(\mathbf{x}, y) d\mathbf{x} dy. \tag{9.14}$$

Because the probability densities are unknown, the expected value of the objective function is replaced by its empirical estimate on the training set. The problem is brought back to that of minimizing the mean squared error objective function:

mean square error

$$J_{\text{MSE}}(f) = \sum_{i=1}^{m}(y_i - f(\mathbf{x}_i))^2 = \sum_{i=1}^{m}(1 - \rho_f(\mathbf{x}_i, y_i))^2, \tag{9.15}$$

where $\rho_f(\mathbf{x}_i, y_i) = y_i f(\mathbf{x}_i)$ are the margin values.

9.3.2.2 Logistic Regression

logistic regression

The logistic regression method, which uses the *maximum likelihood* framework, optimizes a different objective function. It is assumed that $f_{\text{Bayes2}}(\mathbf{x}) = \log[p(\mathbf{x}|y = 1)/p(\mathbf{x}|y = -1)]$ follows a model linear in its parameters: a linear discriminant $f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x})$. One seeks the parameters of $f(\mathbf{x})$ that maximize the likelihood of the training data, assuming that the model is correct. We derive below an objective function for logistic regression in our notations.

Maximizing the likelihood function:

$$\mathcal{L}(\mathbf{w}) = \prod_{i=1}^{m} p(\mathbf{x}_i, y_i|\mathbf{w}) \tag{9.16}$$

is equivalent to minimizing the cross-entropy objective function:[2]

$$J_{\text{Cross\_entropy}}(f) = -\sum_{i=1}^{m} z_i \log(p_i) + (1 - z_i)\log(1 - p_i), \tag{9.17}$$

where $z_i = \frac{y_i+1}{2}$, $p_i = P(y_i = 1|\mathbf{x}_i)$. Using Bayes inversion, we can express p_i as:

$$p_i = \frac{1}{1 + e^{-f_{\text{Bayes2}}(\mathbf{x})}} = \text{logistic}(f_{\text{Bayes2}}(\mathbf{x})) \tag{9.18}$$

where $\text{logistic}(z) = 1/(1 + e^{-z}) = (1 + \tanh(z/2))/2$ is the logistic function. By replacing p_i and z_i by their value in Equation 9.17, the problem can be brought back to that of minimizing:

$$2(J_{\text{Cross\_entropy}}(f) + log(2)) = -\sum_{i=1}^{m}(1 + y_i)\log\left(1 + \tanh\left(\frac{f_{\text{Bayes2}}(\mathbf{x})}{2}\right)\right) \tag{9.19}$$

$$+ (1 - y_i)\log\left(1 - \tanh\left(\frac{f_{\text{Bayes2}}(\mathbf{x})}{2}\right)\right), \tag{9.20}$$

Under the hypothesis that $f_{\text{Bayes2}}(\mathbf{x})$ follows a model linear in its parameters, the objective function becomes:

2. The solution to an optimization problem is not modified by transforming the objective function with a smooth monotically increasing function such as the log and/or removing or adding positive additive or multiplicative constants.

$$J_{\text{Logistic}}(f) = -\sum_{i=1}^{m} \log(1 + \tanh(\rho_f(\mathbf{x}_i, y_i)/2)), \tag{9.21}$$

where $\rho_f(\mathbf{x}_i, y_i) = y_i f(\mathbf{x}_i)$ are the margin values and $f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x})$.
From the axioms of probability, $P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$ always lies between -1 and $+1$. In the MSE approach, the estimate of $P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$ is the linear discriminant itself $f(\mathbf{x}_i)$, which is not limited to taking values between -1 and $+1$.[3] The logistic function approach is consistent with the axioms of probabilities in that its estimate of $P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$, which is $\tanh(f(\mathbf{x})/2)$, always lies between -1 and $+1$.

neural networks

The similarity between the logistic regression approach and the training of a one layer neural network can be noticed. The tanh function is nothing but a sigmoid. Commonly, neural networks are trained with "back-propagation," a gradient descent method using an MSE objective function [Rumelhart et al., 1986b]. The cross-entropy objective function is also sometimes used.

9.3.2.3 Weight Decay

Learning theoretic predictions of classifier performance indicate that if two classifiers have the same training error, the classifier with smallest capacity is more likely to perform better. This is particularly important when problems are very underdetermined, that is when the number of training examples is small compared to the number of parameters to be estimated. Such is the case when a large number of basis functions $\Phi_k(\mathbf{x})$ is used, e.g., in the case of a polynomial classifier.

weight decay

In an effort to choose one solution among many that minimize $J_{\text{MSE}}(f)$ or $J_{\text{Logistic}}(f)$, additional constraints can be added. One popular constraint is to pick the solution that has minimum weight vector norm ($\min \|\mathbf{w}\|$). This is achieved by adding a penalty term to the objective functions, known as "weight decay" term, e.g.,

regularizer

$$J_{\text{WD MSE}}(f) = \sum_{i=1}^{m} (1 - \rho_f(\mathbf{x}_i, y_i))^2 + \lambda \|\mathbf{w}\|^2, \tag{9.22}$$

where λ is a positive constant. The weight decay can be considered as a regularizer.

9.3.3 Maximizing the Minimum Margin

support vector classifier

As with nearest-neighbor classifiers and linear classifiers such as the Perceptron, the support vector classifier approach does not attempt to estimate probabilities.

3. It may seem contradictory that the MSE solution is an approximation to $P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$ and yet is not limited to taking values between -1 and $+1$. However, this approach is valid in the sense that asymptotic convergence is guaranteed.

Instead it provides directly a solution to the classification problem, as do empirical risk minimization methods.

The empirical risk minimization problem, as addressed for instance with the Perceptron method, also suffers from underdetermination. For a large number of basis functions, the training set is likely to be separable. If so, there are usually many linear discriminant functions that perform the separation without error.

In an effort to choose one solution among many, the support vector method advocates to choose the solution that leaves the largest possible margin on both sides of the decision boundary in Φ-space.

optimum margin classifier

There are many equivalent formulations of the optimum margin problem (see the introduction chapter of this book, Chapter 1). One of them resembles the Perceptron formulation (Equation 9.9). In our previous notations one must maximize the smallest possible "margin value" $\rho_f(\mathbf{x}_i, y_i)$, that is minimize the objective function:

$$J_{SVC1}(f) = -\min_{i, 1 \leq i \leq m} \rho_f(\mathbf{x}_i, y_i), \tag{9.23}$$

under the constraints that all examples are well classified and that the weight vector is normalized.

Another version resembles the MSE with weight decay formulation (Equation 9.22). One must minimize the norm of the weight vector under the constraints that the margin values are larger or equal to one. Using the method of Lagrange multipliers, the problem is brought back to that of optimizing the objective function:[4]

$$J_{SVC2}(f) = \|\mathbf{w}\|^2 + \sum_{i=1}^{m} \alpha_i (1 - \rho_f(\mathbf{x}_i, y_i)), \tag{9.24}$$

regularizer

under the constraints that $\alpha_i \geq 0, \forall i, 1 \leq i \leq m$.

In the introduction chapter of this book, Chapter 1, it is explained what type of regularizer is associated with the support vector classifier solution.

9.4 Training Algorithms

There exist a variety of optimization techniques that search for the parameters that minimize the objective functions defined in the previous section. In this section, we focus only on simple gradient descent algorithms. We have several motivations for doing so. We want to provide algorithms that are easy to understand intuitively and easy to implement with a few lines of code. We want to emphasizes connections between the various training algorithms. We refer the advanced user interested in large scale real world problems to algorithms that are more efficient computationally, e.g., use mathematical programming, described elsewhere [Luenberger, 1973].

4. Optimizing means in this case finding the saddle point of $J_{SVC2}(f)$, which is a minimum with respect to $\mathbf{w}$ and a maximimum with respect to α_i.

9.4.1 Gradient Descent

gradient descent

The technique underlying all the algorithms presented in this section is called "gradient descent." Consider an objective function $J(\mathbf{w})$ of some parameters $\mathbf{w}$. Let $\nabla_{\mathbf{w}}$ be the gradient operator:

$$\nabla_{\mathbf{w}} = \left[\frac{\partial}{\partial w_1}, \ldots, \frac{\partial}{\partial w_N}\right]^t. \tag{9.25}$$

At the solution of the minimization problem, we have:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbf{0}. \tag{9.26}$$

The gradient descent technique consists in iteratively approaching the solution by making small steps in steepest direction on the slope of the objective function, as given by the negative gradient:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} J(\mathbf{w}), \tag{9.27}$$

where η is a positive value, called the "learning rate."

stochastic gradient

We distinguish two variants of gradient descent procedures: true gradient and single sample gradient. The true gradient (or batch gradient) method is the method described above. The single sample gradient method (also sometimes called "on-line" or "stochastic" gradient) uses the gradient computed on the cost incurred by a single sample; it updates one sample at a time with this "local" gradient value. We will describe true gradient methods, from which single sample gradient methods can trivially be inferred. Single sample methods are often preferred in practice because of simplicity and speed of convergence.

9.4.2 Algorithms for Linearly Separable Training Sets

Perceptron algorithm

If the training set is linearly separable in Φ-space, it can be shown that minimizing the Perceptron objective function with gradient descent yields a linear discriminant function with zero training error in a finite number fixed size steps (see the introduction chapter of this book, Chapter 1).

We first use the primal expression of the linear discriminant: $f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x})$. The gradient of the objective function $J_{\text{Perceptron}}(f)$ (Equation 9.9) is given by:

$$\nabla_{\mathbf{w}} J_{\text{Perceptron}}(f) = -\sum_{i \in \mathcal{M}} y_i \Phi(\mathbf{x}_i), \tag{9.28}$$

where $\mathcal{M} = \{i, i = 1, \ldots, m : g(\mathbf{x}_i) = \text{sgn}\,(f(\mathbf{x}_i)) \neq y_i\}$ is the set of misclassified examples. From this expression, we can derive algorithm 9.1. By substituting the dual expression of the linear discriminant $f(\mathbf{x}) = \sum_{j=1}^{m} y_j \alpha_j k(\mathbf{x}, \mathbf{x}_j)$, one obtains an alternative dual version. Note that the increments on the αs are of fixed size. This algorithm differs from performing gradient descent in $\boldsymbol{\alpha}$-space directly.

The dual Perceptron algorithm has been known for a long time [Aizerman et al., 1964]. It is a simple way of training kernel classifiers, which is proven to converge in a finite number of steps, as per the Perceptron convergence theorem.

Algorithm 9.1 : Primal and Dual Perceptron

Arguments: Training sample, $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathcal{X}$, $Y = \{y_1, \ldots, y_m\} \subset \{\pm 1\}$.
Returns: Decision function g, parametrized with
primal parameters $\mathbf{w}$ (dim. N Φ-space coordinate weight vector)
or dual parameters $\boldsymbol{\alpha}$ (dim. m sample weight vector).

Function Perceptron(X, Y)
  Initialize arbitrarily, e.g., $\mathbf{w} = \sum_i y_i \Phi(\mathbf{x}_i)$ or $\boldsymbol{\alpha} = 1$.
  repeat
    for all i from $i = 1, \ldots, m$
      Compute the discriminant function values:
      $f(\mathbf{x}_i) = \mathbf{w} \cdot \Phi(\mathbf{x}_i) = \sum_j y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$.
      Compute the margin values: $\rho_f(\mathbf{x}_i, y_i) = y_i f(\mathbf{x}_i)$.
    endfor
    Update on the set of misclassified training examples,
    $\mathcal{M} = \{i, i = 1, \ldots, m : \rho_f(\mathbf{x}_i, y_i) < 0\}$:
    for all $i \in \mathcal{M}$
      primal parameters $\mathbf{w} \leftarrow \mathbf{w} + y_i \Phi(\mathbf{x}_i)$, or
      dual parameter $\alpha_i \leftarrow \alpha_i + 1$.
    endfor
  until all examples are well classified: $\rho_f(\mathbf{x}_i, y_i) > 0$.
  return $g : \mathbf{x} \mapsto \operatorname{sgn}(\mathbf{w} \cdot \Phi(\mathbf{x})) = \operatorname{sgn}(\sum_j y_j \alpha_j k(\mathbf{x}, \mathbf{x}_j))$.
end

Optimum margin Perceptron algorithm

Since the 1992 paper of Boser et al. [Boser et al., 1992], support vector classifiers are often associated with quadratic programming (QP); because of the perceived complexity of QP, this method has had a relatively small impact among pattern recognition practitioners. While we still recommend using QP for large scale real world problems, we present here an alternate algorithm (Table 9.2), which is very simple, and converges to the same solution asymptotically, for $\theta \to \infty$ [Krauth and Mézard, 1987].

Minover

This algorithm closely resembles the Perceptron algorithm. At each iteration, instead of updating the weight vector with the sum of the misclassified examples, one updates with a single example: the example with smallest "margin value." The name of the algorithm is "Minover," for minimum overlap. The concept originated in the physics community where overlap is a synonym for margin value.

Algorithm 9.2 : Primal and Dual Optimum Margin Perceptron (Minover)

Arguments: Training sample, $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathcal{X}$, $Y = \{y_1, \ldots, y_m\} \subset \{\pm 1\}$; stopping criterion, $\theta \in \mathbb{R}^+$.
Returns: Decision function g, parametrized with primal parameters $\mathbf{w}$ (dim. N Φ-space coordinate weight vector) or dual parameters $\boldsymbol{\alpha}$ (dim. m sample weight vector).

Function Minover(X, Y, θ)
  Initialize arbitrarily, e.g., $\mathbf{w} = \sum_i y_i \Phi(\mathbf{x}_i)$ or $\boldsymbol{\alpha} = 1$.
  repeat
    for all i from $i = 1, \ldots, m$
      Compute the discriminant function values: $f(\mathbf{x}_i) = \mathbf{w} \cdot \Phi(\mathbf{x}_i) = \sum_j y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$.
      Compute the margin values: $\rho_f(\mathbf{x}_i, y_i) = y_i f(\mathbf{x}_i)$.
    endfor
    Find the training example $(\mathbf{x}^c, y^c)$, which is ''worst classified,'' i.e., has smallest margin value: $(\mathbf{x}^c, y^c) = \operatorname{argmin}_i \rho_f(\mathbf{x}_i, y_i)$.
    Update primal parameters $\mathbf{w} \leftarrow \mathbf{w} + y^c \Phi(\mathbf{x}^c)$, or increment dual parameter $\alpha^c \leftarrow \alpha^c + 1$.
  until $\rho_f(\mathbf{x}^c, y^c) > \theta$.
  return $g : \mathbf{x} \mapsto \operatorname{sgn}(\mathbf{w} \cdot \Phi(\mathbf{x})) = \operatorname{sgn}(\sum_j y_j \alpha_j k(\mathbf{x}, \mathbf{x}_j))$.
end

Both the Perceptron and the Minover algorithms are independent on the learning rate. The norm of the weight vector increases during learning. One can optionally normalize the weight vector at the end of the training procedure.

Support vector classifiers draw their name from the fact that their discriminant function is a function only of a small number of training examples, called support vectors. Those are the examples with smallest margin value, which are closest to the decision boundary in Φ-space.

informative patterns

The learning mechanism of the Minover algorithm reveals an important aspect of support vectors. Support vectors used to be called "informative patterns." Indeed, in the process of learning, the weight vector is updated only with those patterns that are hardest to predict (have smallest margin value). In the information theoretic sense, the examples that are least predictable are the most informative.

9.4.3 Algorithms for Non Linearly Separable Training Sets

As mentioned in Section 9.2.9, a non linearly separable training set may indicate that the selection of basis functions is not adequate or that the capacity is insufficient. This can be remedied by changing the basis functions (or the kernel) and/or increasing the number of basis functions (or the kernel parameter) to increase the classifier capacity. But it may also indicate overlapping classes or "noisy" data (meaningless or mislabeled examples), in which case it may be detrimental to in-

crease the capacity. Also, we may want to reduce the capacity at the expense of introducing some training error in the hope of obtaining a better prediction error. In these last cases, we need learning algorithms that can converge to a solution with non zero training error.

Although the Perceptron algorithm does not converge in the non-separable case, there exist algorithms (e.g., linear programming) that minimize the Perceptron objective function even in the non-separable case.

Pseudo-inverse and least mean square

Training a linear discriminant function can be thought of as solving a system of linear inequalities:

$$y_i(\mathbf{w} \cdot \Phi(x_i)) > 0, \ i = 1, \ldots, m$$

Traditionally, people have often replaced this problem by that of solving a system of equations:

$$y_i(\mathbf{w} \cdot \Phi(x_i)) = 1, \ i = 1, \ldots, m$$

that provides acceptable solutions in both the separable and non separable case, albeit at the expense of making eventually errors on the training examples in the separable case.

The treatment of the problem is simplified by introducing matrix notations:

$$\mathbf{w}^t \mathbf{\Phi} = \mathbf{y}^t$$

where $\mathbf{w}$ is a $(N, 1)$ matrix, $\mathbf{\Phi} = [\Phi(x_1), \ldots, \Phi(x_m)]$ is a (N, m) matrix, and $\mathbf{y}$ is a $(m, 1)$ matrix. If $\mathbf{\Phi}$ is invertible, the solution can be computed as: $\mathbf{w}^t = \mathbf{y}^t \mathbf{\Phi}^{-1}$. When $\mathbf{\Phi}$ is rectangular, the system is either under or over determined. But one can always seek the best solution in the least square sense, which is given by:

$$\mathbf{w}^t = \mathbf{y}^t \mathbf{\Phi}^+$$

pseudo-inverse

where $\mathbf{\Phi}^+$ is the Moore-Penrose pseudo-inverse, for which many computational algorithms exist [Albert, 1972].

It can be shown that the pseudo-inverse solution minimizes $J_{\text{MSE}}(f)$ (Equation 9.15). Moreover, in the case where the system is underdetermined ($N = dim(\mathbf{w}) > m$) the pseudo-inverse solution is the solution of minimum norm. It is the minimum of $J_{\text{WD MSE}}(f)$ (Equation 9.22), for $\lambda \to 0$.

It is also possible to minimize the mean squared error $J_{\text{MSE}}(f)$ with a gradient descent method. The gradient of $J_{\text{MSE}}(f)$ is:

$$\nabla_{\mathbf{w}} J_{\text{MSE}}(f) = -2 \sum_{i=1}^{m} y_i (1 - \rho_f(\mathbf{x}_i, y_i)) \Phi(\mathbf{x}_i).$$

The weights must be initialized to zero to converge to the solution with minimum norm, the learning rate must decrease with the number of iterations, e.g., $\eta(t) = 1/t$.

Algorithm 9.3 : Primal and Dual Gradient Descent

Arguments: Training sample, $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathcal{X}$, $Y = \{y_1, \ldots, y_m\} \subset \{\pm 1\}$.
learning rate $\eta(t) \in \mathbb{R}^+$,
stopping criterion, $\theta \in \mathbb{R}^+$ and $\tau \in \mathbb{N}$.
Returns: Decision function g, parametrized with
primal parameters $\mathbf{w}$ (dim. N Φ-space coordinate weight vector)
or dual parameters $\boldsymbol{\alpha}$ (dim. m sample weight vector).

Function Gradient$(X, Y, \eta(t), \theta)$
  Initialize $\mathbf{w} = 0$ or $\boldsymbol{\alpha} = 0$ and $t = 0$.
  repeat
    for all i from $i = 1, \ldots, m$
      Compute the discriminant function values:
      $f(\mathbf{x}_i) = \mathbf{w} \cdot \Phi(\mathbf{x}_i) = \sum_j y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$.
      Compute the squashed margin values: $\sigma(\rho_f(\mathbf{x}_i, y_i)) = \sigma(y_i f(\mathbf{x}_i))$.
    endfor
    Update on all training examples,
    for all i from $i = 1, \ldots, m$
      primal parameters $\mathbf{w} \leftarrow \mathbf{w} + \eta(t) y_i [1 - \sigma(\rho_f(\mathbf{x}_i, y_i))] \Phi(\mathbf{x}_i)$, or
      dual parameter $\alpha_i \leftarrow \alpha_i + \eta(t)[1 - \sigma(\rho_f(\mathbf{x}_i, y_i))]$.
    endfor
    Increment t: $t \leftarrow t + 1$.
  until empirical objective function is small enough $J(f) < \theta$ or
    maximum number of iterations is exceeded $t > \tau$.
  return $g : \mathbf{x} \mapsto \operatorname{sgn}(\mathbf{w} \cdot \Phi(\mathbf{x})) = \operatorname{sgn}(\sum_j y_j \alpha_j k(\mathbf{x}, \mathbf{x}_j))$.
end

Depending on the choice of the squashing function, one obtains various algorithms. (1) Perceptron: $\sigma(z) = \operatorname{sgn}(z)$. (2) MSE: $\sigma(z) = z$. (3) Logistic regression: $\sigma(z) = \tanh(z/2)$. (4) Neural soft margin: $\sigma(z) = 1$, if $z < -1$; $\sigma(z) = z$, if $-1 \leq z \leq 1$; $\sigma(z) = 1$, if $z > 1$.

This algorithm highlights the fact, which is not readily apparent in the pseudo-inverse solution, that the MSE solution is a weighted combination of the training patterns. As such, it admits a dual version. Indeed, the Moore-Penrose pseudo-inverse has the following property[5] [Albert, 1972]:

$$\boldsymbol{\Phi}^+ = (\boldsymbol{\Phi}^t \boldsymbol{\Phi})^+ \boldsymbol{\Phi}^t = \boldsymbol{\Phi}^t (\boldsymbol{\Phi} \boldsymbol{\Phi}^t)^+$$

Hence the MSE linear discriminant function has dual forms:

$$f(\mathbf{x}) = \mathbf{w}^t \Phi(\mathbf{x}) = \mathbf{y}^t \boldsymbol{\Phi}^+ \Phi(\mathbf{x}) = \mathbf{y}^t (\boldsymbol{\Phi}^t \boldsymbol{\Phi})^+ \boldsymbol{\Phi}^t \Phi(\mathbf{x}) \tag{9.29}$$

$$= \mathbf{y}^t K^+ \boldsymbol{\kappa}(\mathbf{x}) = \boldsymbol{\beta}^t \boldsymbol{\kappa}(\mathbf{x}) = \sum_{i=1}^{n} y_i \alpha_i k(\mathbf{x}, \mathbf{x}_i) \tag{9.30}$$

5. $(\boldsymbol{\Phi}^t \boldsymbol{\Phi})^+ = (\boldsymbol{\Phi}^t \boldsymbol{\Phi})^{-1}$ if the columns of $\boldsymbol{\Phi}$ are linearly independent and $(\boldsymbol{\Phi} \boldsymbol{\Phi}^t)^+ = (\boldsymbol{\Phi} \boldsymbol{\Phi}^t)^{-1}$ if the rows of $\boldsymbol{\Phi}$ are linearly independent. This provides one way of computing the pseudo-inverse.

where $K = \mathbf{\Phi}^t\mathbf{\Phi} = [k(\mathbf{x}_i, \mathbf{x}_j)]_{1\leq i\leq m, 1\leq j\leq m}$ is a (m, m) matrix to be pseudo-inverted, $\boldsymbol{\kappa}(\mathbf{x}) = \mathbf{\Phi}^t\Phi(\mathbf{x}) = [k(\mathbf{x}, \mathbf{x}_i)]_{1\leq i\leq m}$ is a $(m, 1)$ matrix, and $\boldsymbol{\beta}^t = \mathbf{y}^t K^+$ is a $(1, m)$ matrix of elements $y_i\alpha_i$.

Interestingly, the pseudo-inverse/MSE solution coincides with the optimum margin solution when all the training examples are support vectors.

Logistic regression

Another very similar algorithm is obtained by minimizing $J_{\text{logistic}}(f)$. The gradient of $J_{\text{logistic}}(f)$ with respect to $\mathbf{w}$ is:

$$\nabla_{\mathbf{w}} J_{\text{logistic}}(f) = -\frac{1}{2}\sum_{i=1}^{m} y_i(1 - \tanh(\rho_f(\mathbf{x}_i, y_i)/2))\Phi(\mathbf{x}_i).$$

back-propagation

As shown in Algorithm 9.3 the only change in the algorithm is to pass the margin values through a squashing function.

Note that, unlike the back-propagation algorithm that trains one layer neural networks with gradient descent using the MSE objective function, the weight update is not multiplied by the derivative of the sigmoid (tanh) function.

A Minover algorithm for soft margin classifier

negative margin

One may wonder whether it is possible to obtain a support vector classifier for non separable training data, by extending the idea of an optimum margin.

The first idea that comes to mind is to minimize the negative margin, instead of maximizing the positive margin. There are a number of reasons why this is not a good idea. First the solution may not be unique. It is easy to construct examples for which there exist several equivalent negative margin solutions (see Figure 9.4.3) [Lambert, 1969]. Second, the negative margin is solely defined by misclassified examples, which may be "bad" examples to rely on.

Another possibility is to keep maximizing the positive margin, but allow that a number of training examples be misclassified, with a certain penalty for each misclassified example. This is the idea behind the soft margin algorithm [Cortes and Vapnik, 1995] and ν-SVC [Schölkopf et al., 1998c] .

soft margin

The quadratic programming formulation of the soft margin algorithm and ν-SVC are very similar to that of the regular maximum margin algorithm. For the soft margin algorithm, there is only a set of additional constraints on the Lagrange multipliers: $\alpha_i \leq C$, where C is a positive constant. One can trivially extend the Minover algorithm to compute the soft margin solution (Algorithm 9.4).[6]

While the original Minover algorithm insists on trying to learn examples that have negative margin values and cannot be well classified, the soft Minover algorithm gives them up after a while and continues with the remaining examples.

---

6. ν-SVC provides a more explicit control over the number of non-marginal support vectors, but it does not lend itself to a simple extention of the Minover algorithm.

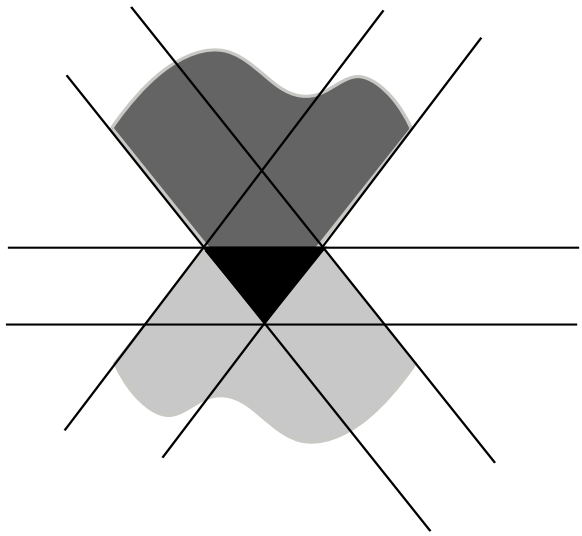


Figure 9.2 Negative margin. Examples can be constructed that have several equivalent negative margin solutions.

Neural networks and large margin classifiers

In the separable training set case, if all the training examples are support vectors, the pseudo-inverse/MSE solution is also the maximum margin solution. This is not the case in general.

The Kernel-Adatron method [Frieß et al., 1998] is a simple modification of the MSE update (Algorithm 9.3 (2)), for which α_i is replaced by zero if it becomes negative. It can be shown to converge to the maximum margin solution. The soft margin constraints $\alpha_i \leq C$ can similarly be enforced by replacing α_i by C if it goes overbound.

One of the effects of the squashing function in the logistic regression is to limit the influence of very well classified examples (with large positive margins). Another effect is to limit the influence of misclassified examples (with large negative margins). This is even emphasized in the back-propagation algorithm, because the weight update is multiplied by the derivative of the sigmoid (tanh) function.

These two effects are similar in spirit to the constraints imposed on α_i by the soft margin algorithm: $0 \leq \alpha_i \leq C$.

If the tanh function is replaced by a piecewise linear squashing function ($\sigma(z) = -1$, if $z < -1$; $\sigma(z) = z$, if $-1 \leq z \leq 1$; $\sigma(z) = 1$, if $z > 1$), another algorithm that computes a large margin solution is obtained. If an example is very well classified ($\rho_f(\mathbf{x}_i, y_i) \geq 1$), it does not contribute to the weight update. If an example is very misclassified ($\rho_f(\mathbf{x}_i, y_i) \leq -1$), it contributes a fixed maximum increment.

Algorithm 9.4 : Primal and Dual Soft Margin Perceptron (SoftMinover)

Arguments: Training sample, $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathcal{X}$, $Y = \{y_1, \ldots, y_m\} \subset \{\pm 1\}$;
stopping criterion, $\theta \in \mathbb{R}^+$;
soft margin regularization constant, $C \in \mathbb{R}^+$.
Returns: Decision function g, parametrized with
primal parameters $\mathbf{w}$ (dim. N Φ-space coordinate weight vector)
or dual parameters $\boldsymbol{\alpha}$ (dim. m sample weight vector).

Function SoftMinover(X, Y, θ, C)
  Initialize: $\mathbf{w} = \sum_i y_i \Phi(\mathbf{x}_i)$ and $\boldsymbol{\alpha} = 1$.
  repeat
    for all i from $i = 1, \ldots, m$
      Compute the discriminant function values:
      $f(\mathbf{x}_i) = \mathbf{w} \cdot \Phi(\mathbf{x}_i) = \sum_j y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$.
      Compute the margin values: $\rho_f(\mathbf{x}_i, y_i) = y_i f(\mathbf{x}_i)$.
    endfor
    Find the training example $(\mathbf{x}^c, y^c)$, which is ''worst classified,''
    i.e., has smallest margin value: $(\mathbf{x}^c, y^c) = \operatorname{argmin}_i \rho_f(\mathbf{x}_i, y_i)$.
    if the corresponding α^c goes over bound: $\alpha^c > C$
      Take the example $(\mathbf{x}^c, y^c)$ out of the training set.
    else
      Update primal parameters $\mathbf{w} \leftarrow \mathbf{w} + y^c \Phi(\mathbf{x}^c)$, and
      increment dual parameter $\alpha^c \leftarrow \alpha^c + 1$.
    endif
  until $\rho_f(\mathbf{x}^c, y^c) > \theta$ or the training set is empty.
  return $g : \mathbf{x} \mapsto \operatorname{sgn}(\mathbf{w} \cdot \Phi(\mathbf{x})) = \operatorname{sgn}(\sum_j y_j \alpha_j k(\mathbf{x}, \mathbf{x}_j))$.
end

In the process of learning, the weight vector increases and more and more examples having margin values greater than one become inactive. The algorithm progressively focuses on the marginal examples. If the algorithm is initialized with zero weights, in the separable training set case, it converges to the MSE solution of minimum norm computed on the marginal examples only. This is precisely the maximum margin solution.

neural
soft margin

In the non-separable case, the algorithm ends up focusing on the misclassified examples. The backpropagation variant that multiplies the update by the derivative of the squashing function may be preferable. If an example is very misclassified ($\rho_f(\mathbf{x}_i, y_i) \leq -1$), it does not contribute to the weight update. This implements a kind of soft margin algorithm (9.3) (4) that we refer to as "neural soft margin."

9.5 Which Linear Discriminant?

While there is academic value in comparing methods, in practice it is often difficult to know which linear discriminant method is best suited to the classification problem at hand. As we have already pointed out, each approach has strengths and weaknesses, and may be optimum on some criterion. Although recent results — both theoretical and experimental — argue in favor of SVCs from the point of view of generalization (see the introduction chapter of this book, Chapter 1), occasionally other practical considerations may be important, as we now consider.

9.5.1 Feature Selection and Computational Burden

A fact that is not readily apparent from our presentation is that most linear discriminant methods suffer from intrinsic computational limitations. As discussed in Section 9.2, for a good choice of the basis functions, linear discriminants yield classification functions that can approximate the Bayes optimum decision function arbitrarily well. One approach may be to select a polynomial classifier of high order, that is, to use all the monomials up to a certain order as basis functions. Training such a classifier in the primal representation using traditional methods is impractical since the number of parameters to estimate is too large. In the dual representation, training is possible only if the number of training examples is not too large. Traditionally, this problem is addressed with various auxiliary techniques of feature selection (in primal space) or example selection (in dual space).

The advantage of SVCs is that they avoid this difficult feature selection step. Computationally, one can capitalize on the fact that the solution is a function only of the SVs, a small subset of the training patterns, where learning relies on quadratic programming in the dual space [Boser et al., 1992]. In particular, SVCs perform an automatic feature selection and example selection via the selection of SVs.

9.5.2 Probabilistic Interpretation of the Scores

So far, we have only considered discriminant functions as a means to a final classification decision; only the sign of $f(\mathbf{x})$ matters for this purpose. However, when a classifier is integrated as part of a larger system, the analog value of $f(\mathbf{x})$ itself often provides valuable information. For example, consider the design of a recognition system for zipcodes, unsegmented strings of digits. A single digit classifier may be integrated into such a zipcode recognition system in the following way. The overall system tries several heuristic segmentations of the input string; for each such "tentative" segmentation, the various segments are submitted to the digit classifier. Here, it is important that the digit classifier return a score or a confidence value rather than a simple digit classification so that an overall score for the whole string for various tentative segmentations can be computed. The highest such score is selected, yielding the chosen zipcode classification.

In other problems, it is convenient to use the value of $|f(\mathbf{x})|$ as a confidence value on which a threshold can be set in order to reject test examples that are ambiguous or meaningless, rather than making a classification error. The question is therefore how well $f(\mathbf{x})$ serves as a score for various linear discriminants. While it is not possible to answer that question for every situation, it is worth mentioning that both the MSE and the logistic regression methods approximate the optimum Bayes discriminant; their scores can therefore readily be interpreted in terms of probability estimates. We generally favor the logistic regression estimate because $f(\mathbf{x})$ provides an estimate of $f_{\text{Bayes1}} = P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$, which belongs to the $[-1, +1]$ interval while the MSE estimate does not.

The score provided by SVCs has no direct probabilistic interpretation. Nevertheless, there is an analogy between the tanh squashing function of logistic regression and the piecewise linear squashing function of the "neural soft margin." This suggests that squashing the score of the SVC classifier with a tanh may be a good idea, if a heuristic probabilistic interpretation needs to be made. If additional data is available, it is also possible to add a postprocessor that remaps the scores to probabilities [Schürmann, 1996], e.g., by fitting the parameters of a sigmoid $\tanh(\mathbf{a}\cdot\mathbf{x}+b)$ (see Chapter 5).

9.5.3 Robustness, Regularization, Good and Bad Outliers

Another important practical question is the treatment of outliers. In the context of SVCs, outliers may be defined as a training examples with a small margin value. On one hand, SVCs tend to emphasize outliers, which are often found among the support vectors. On the other hand, robust statistics methods derived from MSE training and logistic regression go in the opposite direction and attempt to de-emphasize the importance of outliers [Hampel et al., 1986]. Increasing a regularization penalty is an effective method of reducing the importance of outliers. The soft margin method [Cortes and Vapnik, 1995] and ν-SVC [Schölkopf et al., 1998c] reconcile SVCs with robust statistics by limiting the influence of the worst outliers. For more precise robust statistics claims, see [Schölkopf et al., 1998c].

There is no single good nor single wrong method; nevertheless there are good and bad outliers. Good outliers are very informative: *ambiguous patterns* that help defining crisply the decision boundary or *rare patterns* that help defining the decision boundary in regions that are not densely populated. Bad outliers may be "informative" in the information theoretic sense — i.e., hard to predict — but very "non-informative" in practice, i.e., not useful. Such outliers include *mislabeled* and *meaningless* patterns. These bad outliers correspond to errors introduced in the data and nearly always reduce the accuracy of the final classifier.

The problem of robust methods is that they de-emphasize outliers regardless of whether they are informative of not. On the contrary, SVCs emphasize them equally blindly. The solution is called "data cleaning." Since cleaning by verifying all data entries is tedious, SVCs can be put to work in a bootstrap method. A first classifier is trained with unclean data. Then, its support vectors (ranked in order of

decreasing αs) are examined and the "bad outliers" eliminated. The overall process is iterated until no such "bad" outliers remain [Guyon et al., 1996].

9.6 Conclusion

This chapter has explored the commonalties and relationships between linear discriminant functions and support vector classifiers. Naturally, we have not exhausted this subject, but we have described the following connections between "classical" linear discriminant and SVC: Similarities in the objective functions, which typically exhibit a tradeoff between minimizing the number of training errors and minimizing classifier complexity. Similarities in gradient descent algorithms, whose parameter update is proportional to $[1 - \sigma(\rho_f(\mathbf{x}_i, y_i))]$, where the margin value $\rho_f(\mathbf{x}_i, y_i))$ is proportional to the distance of the training example to the decision boundary, and $\sigma(\cdot)$ is a squashing function such as a sigmoid. Similarities in the way duality can be exploited during training to simplify computations. Similarities in the probabilistic interpretation of the scores, in particular how the function $\sigma(f(\mathbf{x}_i))$ can be thought of as an approximation to the Bayes optimum discriminant function $f_{\text{Bayes1}} = P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$. Of course, by emphasizing similarities we do not mean to minimize the differences and benefits of SVCs over other linear discriminant techniques, as described by many others (see the introduction chapter of this book, Chapter 1). For instance SVCs draw their unique properties from the existence of support vectors and one can capitalize on the fact that the solution is a function only of the support vectors, a small subset of the training patterns. Quadratic programming can be used to find the optimum margin solution in dual space [Boser et al., 1992]. Theoretical results and experimental evidence show that SVCs draw advantages from their unique use of support vectors. Other subjects — robustness, regularization good and bad outliers — deserve more attention but go beyond the scope of this chapter.

Acknowledgments

Isabelle Guyon would like to thank Léon Personnaz for his initial impulse to the ideas of using duality in pseudo-inverse and other linear discriminant algorithms, and Léon Bottou for pointing out the "neural soft margin" algorithm.

10 Regularization Networks and Support Vector Machines

Theodoros Evgeniou
Center for Biological and Computational Learning, MIT
45 Carleton Street E25-201
Cambridge, MA 02142, USA
theos@ai.mit.edu

Massimiliano Pontil
Center for Biological and Computational Learning, MIT
45 Carleton Street E25-201
Cambridge, MA 02142, USA
pontil@ai.mit.edu

Tomaso Poggio
Center for Biological and Computational Learning, MIT
45 Carleton Street E25-201
Cambridge, MA 02142, USA
tp@ai.mit.edu

Regularization Networks and Support Vector Machines are techniques for solving certain problems of learning from examples – in particular the regression problem of approximating a multivariate function from sparse data. We present both formulations in a unified framework, namely in the context of Vapnik's theory of statistical learning which provides a general foundation for the learning problem, combining functional analysis and statistics.[1]

1. This chapter is a short version of a paper submitted to Advances in Computational Mathematics, which is available as AI-Memo by anonymous ftp at the URL ftp://publications.ai.mit.edu/1500-1999/AIM-1654.ps.

10.1 Introduction

The purpose of this chapter is to present a theoretical framework for the problem of learning from examples. Learning from examples can be regarded as the regression problem of approximating a multivariate function from sparse data – and we will take this point of view here.[2]

The problem of approximating a function from sparse data is ill-posed and a classical way to solve it is regularization theory [Tikhonov and Arsenin, 1977, Bertero, 1986, Bertero et al., 1988, Wahba, 1990]. Standard regularization theory, as we will consider here, formulates the regression problem as a variational problem of finding the function f that minimizes the functional

$$\min_{f \in F} R_{\text{reg}}[f] = \frac{1}{m} \sum_{i=1} (y_i - f(\mathbf{x}_i))^2 + \lambda \|f\|_k^2 \tag{10.1}$$

where $\|f\|_k^2$ is a norm in a Reproducing Kernel Hilbert Space (RKHS) F defined by the positive definite function k, m is the number of data points or examples (the m pairs $(\mathbf{x}_i, y_i)$) and λ is the regularization parameter. Under rather general conditions the solution of equation (10.1) is

$$f(\mathbf{x}) = \sum_{i=1} \alpha_i k(\mathbf{x}, \mathbf{x}_i). \tag{10.2}$$

Until now the functionals of standard regularization have lacked a rigorous justification for a finite set of training data. Their formulation is based on functional analysis arguments which rely on asymptotic results and do not consider finite data sets.[3] Regularization is the approach we have taken in earlier work on learning [Poggio and Girosi, 1989, Girosi et al., 1995, Powell, 1992]. The seminal work of Vapnik [1979, 1995, 1998] has now set the foundations for a more general theory that justifies regularization functionals for learning from finite sets and can be used to extend considerably the classical framework of regularization, effectively marrying a functional analysis perspective with modern advances in the theory of probability and statistics. The basic idea of Vapnik's theory is closely related to regularization: for a finite set of training examples the search for the best model or approximating function has to be constrained to an appropriately "small" hypothesis space (which can also be thought of as a space of machines or models or network architectures). If the space is too large, models can be found which will fit exactly the data but will have a poor generalization performance, that is poor predictive capability on new data. Vapnik's theory characterizes and formalizes these concepts in terms of the

2. There is a large literature on the subject: useful reviews are [Haykin, 1994, Cherkassky and Mulier, 1998, Girosi et al., 1995, Vapnik, 1998] and references therein.
3. The method of quasi-solutions of Ivanov [1976] and the equivalent Tikhonov's regularization technique were developed to solve ill-posed problems of the type $Af = B$, where A is a (linear) operator, f is the desired solution in a metric space E_1, and B are the "data" in a metric space E_2.

capacity of a set of functions and *capacity control* depending on the training data: for instance, for a small training set the capacity of the function space in which f is sought has to be small whereas it can increase with a larger training set. As we will see later in the case of regularization, a form of capacity control leads to choosing an optimal λ in equation (10.1) for a given set of data. A key part of the theory is to define and bound the capacity of a set of functions.

Thus the key and somewhat novel theme of this review is a) to describe a unified framework for several learning techniques for finite training sets and b) to justify them in terms of statistical learning theory. We will consider functionals of the form

$$R_{\text{reg}}[f] = \frac{1}{m}\sum_{i=1} c(\mathbf{x}_i, y_i, f(\mathbf{x})) + \lambda \|f\|_k^2, \tag{10.3}$$

where $c(\cdot,\cdot,\cdot)$ is a *loss function*. We will describe how standard regularization and Support Vector Machines [Vapnik, 1998] for both regression and classification correspond to the minimization of R_{reg} in (10.3) for different choices of c:

- Standard (L_2) Regularization Networks (RN)

$$c(\mathbf{x}_i, y_i, f(\mathbf{x})) = (y_i - f(\mathbf{x}_i))^2 \tag{10.4}$$

- Support Vector Machines Regression (SVMR)

$$c(\mathbf{x}_i, y_i, f(\mathbf{x})) = |y_i - f(\mathbf{x}_i)|_\epsilon \tag{10.5}$$

- Support Vector Machines Classification (SVMC)

$$c(\mathbf{x}_i, y_i, f(\mathbf{x})) = \theta(1 - y_i f(\mathbf{x}_i))(1 - y_i f(\mathbf{x}_i)) \tag{10.6}$$

where $|\cdot|_\epsilon$ is Vapnik's epsilon-insensitive norm (see later), $\theta(\cdot)$ is the Heaviside function and y_i is a real number in RN and SVMR, whereas it takes values $-1, 1$ in SVMC. Loss function (10.6) is also called the *soft margin* loss function. For SVMC, we will also discuss two other loss functions:

- The *hard margin* loss function:

$$c(\mathbf{x}_i, y_i, f(\mathbf{x})) = \theta(1 - y_i f(\mathbf{x}_i)) \tag{10.7}$$

- The *misclassification* loss function:

$$c(\mathbf{x}_i, y_i, f(\mathbf{x})) = \theta(-y_i f(\mathbf{x}_i)) \tag{10.8}$$

For classification one should minimize (10.8) (or (10.7)), but in practice other loss functions, such as the soft margin one (10.6) [Cortes and Vapnik, 1995, Vapnik, 1995], are used. We discuss this issue further in Section 10.5.

The minimizer of (10.3) using the three loss functions has the same general form (10.2) (or $f(\mathbf{x}) = \sum_{i=1} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b$, see later) but interestingly different properties. In this review we will show how different learning techniques based on the minimization of functionals of the form of R_{reg} in (10.3) can be justified for a few choices of $c(\cdot,\cdot,\cdot)$ using a slight extension of the tools and results of Vapnik's statistical learning theory. In Section 10.2 we outline the main results in the theory

of statistical learning and in particular Structural Risk Minimization – the technique suggested by Vapnik to solve the problem of capacity control in learning from "small" training sets. At the end of the section we will outline a technical extension of Vapnik's Structural Risk Minimization framework (SRM). With this extension both RN and Support Vector Machines (SVMs) can be seen within a SRM scheme. In recent years a number of papers claim that SVM cannot be justified in a data-independent SRM framework (i.e., [Shawe-Taylor et al., 1998]). One of the goals of this chapter is to provide such a data-independent SRM framework that justifies SVM as well as RN. After the section on regularization (Section 10.3) we will describe SVMs (Section 10.4). As we saw already, SVMs for regression can be considered as a modification of regularization formulations of the type of (10.1).

Section 10.5 describes in more detail how and why *both* RN and SVM can be justified in terms of SRM, in the sense of Vapnik's theory: the key to capacity control is how to choose λ for a given set of data. Section 10.6 describes a naive maximum a posteriori (MAP) Bayesian interpretation of RNs and of SVMs. It also shows why a formal MAP Bayesian interpretation, though interesting and even useful, may be somewhat misleading. Section 10.7 discusses relations of the regularization and SVM techniques with other representations of functions and signals such as sparse representations from overcomplete dictionaries.

10.2 Overview of Statistical Learning Theory

Statistical learning theory as developed by Vapnik builds on the so-called *empirical risk minimization (ERM)* induction principle. The ERM method consists in using the training data set $X \times Y = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, with $(\mathbf{x}_i, y_i) \in \mathbb{R}^N \times \mathbb{R}$ sampled from an unknown probability distribution $p(\mathbf{x}, y)$, to build a stochastic approximation of the expected risk (see also Section 1.2.1)

$$R(f) := \int_{\mathbb{R}^N \times \mathbb{R}} c(\mathbf{x}, y, f(\mathbf{x}))\, p(\mathbf{x}, y)\, d\mathbf{x}\, dy, \tag{10.9}$$

namely the *empirical risk*:

$$R_{\text{emp}}(f) := \frac{1}{m} \sum_{i=1} c(\mathbf{x}_i, y_i, f(\mathbf{x})). \tag{10.10}$$

The central question of statistical learning theory is whether the expected risk of the minimizer of the empirical risk in a hypothesis space F is close to the expected risk of the minimizer of the expected risk in F, f_0. Notice that the question is not necessarily whether we can find f_0 but whether we can *"imitate"* f_0 in the sense that the expected risk of our solution is close to that of f_0. Formally the theory answers the question of finding under which conditions the method of ERM satisfies:

$$\lim_{m \to \infty} R_{\text{emp}}(\hat{f}_m) = \lim_{m \to \infty} R(\hat{f}_m) = R(f_0) \tag{10.11}$$

in probability (all statements are probabilistic since we start with $p(\mathbf{x}, y)$ on the data), where we note with $\hat{f}_m$ the minimizer of the empirical risk (10.10) in F.

It can be shown (see for example [Vapnik, 1998]) that in order for the limits in eq. (10.11) to hold true in probability, or more precisely, for the empirical risk minimization principle to be *non-trivially consistent* (see [Vapnik, 1998] for a discussion about consistency versus non-trivial consistency), the following *uniform law of large numbers* (which "translates" to *one-sided uniform convergence in probability* of empirical risk to expected risk in F) is a *necessary and sufficient* condition:

$$\lim_{m\to\infty} Pr\left\{\sup_{f\in F}\left(R(f) - R_{\text{emp}}(f)\right) > \epsilon\right\} = 0 \quad \forall\epsilon > 0 \tag{10.12}$$

Intuitively, if F is very "large" then we can always find $\hat{f}_m \in F$ with 0 empirical error. This however does not guarantee that the expected risk of $\hat{f}_m$ is also close to 0, or close to $R(f_0)$.

Typically in the literature the *two-sided uniform convergence in probability*:

$$\lim_{m\to\infty} Pr\left\{\sup_{f\in F}\left|R(f) - R_{\text{emp}}(f)\right| > \epsilon\right\} = 0 \quad \forall\epsilon > 0 \tag{10.13}$$

is considered, which clearly implies (10.12). In this chapter we focus on the stronger two-sided case and note that one can get one-sided uniform convergence with some minor technical changes to the theory. We will not discuss the technical issues involved in the relations between consistency, non-trivial consistency, two-sided and one-sided uniform convergence (a discussion can be found in [Vapnik, 1998]), and from now on we concentrate on the two-sided uniform convergence in probability, which we simply refer to as *uniform convergence*.

The theory of uniform convergence of ERM has been developed in [Vapnik and Chervonenkis, 1971, 1981, 1991, Vapnik, 1979, 1998]. It has also been studied in the context of *empirical processes* [Dudley, 1984, Pollard, 1984, Dudley et al., 1991]. Here we summarize the main results of the theory.

10.2.1 Uniform Convergence and the Vapnik-Chervonenkis Bound

Vapnik and Chervonenkis [1971, 1981] studied under what conditions uniform convergence of the empirical risk to expected risk takes place. The results are formulated in terms of three important quantities that measure the complexity of a set of functions: the *VC entropy*, the *annealed VC entropy*, and the *growth function*. We begin with the definitions of these quantities.

Definition 10.1
Given a probability $p(\mathbf{x}, y)$ over $\mathbb{R}^N \times \mathbb{R}$, the *VC entropy* of a set of functions $\{c(\mathbf{x}, y, f(\mathbf{x})) \ : \ f \in F\}$, on a data set of size m is defined as:

$$H^F(\epsilon; m) \equiv \int_{\mathbb{R}^N \times \mathbb{R}} \ln \mathcal{N}(\epsilon, F, X \times Y) \prod_{i=1} p(\mathbf{x}_i, y_i) d\mathbf{x}_i dy_i \tag{10.14}$$

where $\mathcal{N}(\epsilon, F, X \times Y)$, with $X \times Y = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, is the size of the minimal ϵ-net (see also definition 1.8) of the set:

$$\{q(f; X \times Y) = (c(\mathbf{x}_1, y_1, f(\mathbf{x})), ..., c(\mathbf{x}_m, y_m, f(\mathbf{x}))) : f \in F\} \tag{10.15}$$

under the metric:

$$\begin{aligned} & \ell_{\infty, X \times Y} \left(q(f; X \times Y), q(f'; X \times Y)\right) \\ = & \max_{1 \le i \le m} |c(\mathbf{x}_i, y_i, f(\mathbf{x})) - c(\mathbf{x}_i, y_i, f'(\mathbf{x}))| \end{aligned} \tag{10.16}$$

Definition 10.2
Given a probability $p(\mathbf{x}, y)$ over $\mathbb{R}^N \times \mathbb{R}$, the *annealed VC entropy* of a set of functions $\{c(\mathbf{x}, y, f(\mathbf{x})) \ : \ f \in F\}$, on a data set of size m is defined as:

$$H^F_{\text{ann}}(\epsilon; m) \equiv \ln \int_{\mathbb{R}^N \times \mathbb{R}} \mathcal{N}(\epsilon, F, X \times Y) \prod_{i=1} p(\mathbf{x}_i, y_i) d\mathbf{x}_i dy_i \tag{10.17}$$

Definition 10.3
The *growth function* of a set of functions $\{c(\mathbf{x}, y, f(\mathbf{x})) \ : \ f \in F\}$, on a data set of size m is defined as:

$$G^F(\epsilon; m) \equiv \ln \left(\sup_{X \times Y \in (\mathbb{R}^N \times \mathbb{R})} \mathcal{N}(\epsilon, F, X \times Y) \right) = \ln \mathcal{N}(\epsilon, F, m) \tag{10.18}$$

(where $\mathcal{N}(\epsilon, F, m)$ is as defined in Chapter 1).

Notice that all three quantities are functions of the number of data m and of ϵ, and that clearly:

$$H^F(\epsilon; m) \le H^F_{\text{ann}}(\epsilon; m) \le G^F(\epsilon; m) \ . \tag{10.19}$$

These definitions can easily be extended in the case of *indicator functions*, i.e., functions c taking binary values[4] such as $\{-1, 1\}$, in which case the three quantities do not depend on ϵ for $\epsilon < 1$, since all vectors of the set (10.15) are at the vertices of the hypercube $\{0, 1\}$.

Using these definitions we can now state three important results of statistical learning theory [Vapnik, 1998]:

4. In the case of indicator functions, y is binary, and c is 0 for $f(\mathbf{x}) = y$, 1 otherwise.

- For a given probability distribution $p(\mathbf{x}, y)$:
  1. The necessary and sufficient condition for uniform convergence is that
  $$\lim_{m\to\infty} \frac{H^F(\epsilon; m)}{m} = 0 \quad \forall \epsilon > 0 \tag{10.20}$$
  2. A sufficient condition for *fast asymptotic rate of convergence*[5] is that
  $$\lim_{m\to\infty} \frac{H^F_{\text{ann}}(\epsilon; m)}{m} = 0 \quad \forall \epsilon > 0 \tag{10.21}$$
  It is an open question whether this is also a necessary condition.
- A sufficient condition for distribution *independent* (that is, for any $p(\mathbf{x}, y)$) fast rate of convergence is that

$$\lim_{m\to\infty} \frac{G^F(\epsilon; m)}{m} = 0 \quad \forall \epsilon > 0 \tag{10.22}$$

For indicator functions this is also a necessary condition.

According to statistical learning theory, these three quantities are what one should consider when designing and analyzing learning machines: the VC-entropy and the annealed VC-entropy for an analysis which depends on the probability distribution $p(\mathbf{x}, y)$ of the data, and the growth function for a distribution *independent* analysis. In this chapter we consider only distribution *independent* results, although the reader should keep in mind that distribution dependent results are likely to be important in the future.

Unfortunately the growth function of a set of functions is difficult to compute in practice. So the standard approach in statistical learning theory is to use an upper bound on the growth function which is given using another important quantity, the *VC-dimension* (see definition 1.4), which is another (*looser*) measure of the complexity, *capacity*, of a set of functions (also provides an upper bound on the growth function). In this chapter we concentrate on this quantity, but it is important that the reader keeps in mind that the VC-dimension is in a sense a "weak" measure of complexity of a set of functions, so it typically leads to loose upper bounds on the growth function: in general one is better off, theoretically, using directly the growth function.

The remarkable property of the VC-dimension is that, although as we mentioned it only provides an upper bound to the growth function, in the case of indicator functions, *finiteness of the VC-dimension is a* **necessary and sufficient** *condition for uniform convergence (eq. (10.13))* **independent** *of the underlying distribution* $p(\mathbf{x}, y)$. However, in the case of real valued functions, finiteness of the VC-dimension is *only sufficient* for uniform convergence. Later in this section we will discuss a measure of capacity that provides also necessary conditions.

---

5. This means that for any $m > m_0$ we have that $Pr\{\sup_{f\in F} |R(f) - R_{\text{emp}}(f)| > \epsilon\} < e^{-c\epsilon^2 m}$ for some constant $c > 0$. Intuitively, fast rate is typically needed in practice.

The VC-dimension can be used to get bounds on the expected risk of $\hat{f}_m$.[6] In particular (see introduction), if h is the VC-dimension of a set of functions F, and $A \leq c(\mathbf{x}, y, f(\mathbf{x})) \leq B$, then the following inequality holds with probability $1 - \eta$:

$$\left| R(f_0) - R(\hat{f}_m) \right| \leq 2(B - A)\sqrt{\frac{h \ln \frac{2em}{h} - \ln(\frac{\eta}{4})}{m}} \tag{10.23}$$

Furthermore the following bounds holds with probability $1 - \eta$ uniformly for *all* functions $f \in F$:

$$|R(f) - R_{\text{emp}}(f)| \leq (B - A)\sqrt{\frac{h \ln \frac{2em}{h} - \ln(\frac{\eta}{4})}{m}} \tag{10.24}$$

Inequalities (10.24) and (10.23) suggest a method for achieving good generalization: not only minimize the empirical risk, but instead minimize a combination of the empirical risk and the complexity of the hypothesis space. This observation leads us to the method of *Structural Risk Minimization* that we describe next.

10.2.2 The Method of Structural Risk Minimization

The idea of SRM is to define a nested sequence of hypothesis spaces $F_1 \subset F_2 \subset \ldots \subset F_{n(m)}$ with $n(m)$ a non-decreasing integer function of m, where each hypothesis space F_i has VC-dimension finite and larger than that of all previous sets, i.e., if h_i is the VC-dimension of space F_i, then $h_1 \leq h_2 \leq \ldots \leq h_{n(m)}$. For example F_i could be the set of polynomials of degree i, or a set of splines with i nodes, or some more complicated nonlinear parameterization. For each element F_i of the structure the solution of the learning problem is:

$$\hat{f}_{i,m} = \arg\min_{f \in F_i} R_{\text{emp}}(f) \tag{10.25}$$

Because of the way we define our structure it should be clear that the larger i is the smaller the empirical error of $\hat{f}_{i,m}$ is (since we have greater "flexibility" to fit our training data), but the larger the VC-dimension part (second term) of the right hand side of (10.24) is. Using such a nested sequence of more and more complex hypothesis spaces, the SRM learning technique consists of choosing the space $F_{n^*(m)}$ for which the right hand side of inequality (10.24) is minimized. It can be shown [Vapnik, 1979] that for the chosen solution $\hat{f}_{n^*(m),m}$ inequalities (10.24) and (10.23) hold with probability at least $(1 - \eta)^{n(m)} \approx 1 - n(m)\eta$,[7] where we replace h with $h_{n^*(m)}$, f_0 with the minimizer of the expected risk in $F_{n^*(m)}$, namely $f_{n^*(m)}$, and $\hat{f}_m$ with $\hat{f}_{n^*(m),m}$.

---

6. It is important to note that bounds on the expected risk of $\hat{f}_m$ using the annealed VC-entropy also exist. These are tighter than the VC-dimension ones.

7. We want (10.24) to hold simultaneously for all spaces F_i, since we choose the best $\hat{f}_{i,m}$.

With an appropriate choice of $n(m)$[8] it can be shown that as $m \to \infty$ and $n(m) \to \infty$, the expected risk of the solution of the method approaches in probability the minimum of the expected risk in $F = \bigcup_{i=1}^{\infty} F_i$, call it $R(f_F)$. Moreover, if the (true) target function f_0 belongs to the closure of F, then eq. (10.11) holds in probability (see for example [Vapnik, 1998]).

However, in practice m is finite ("small"), so $n(m)$ is small which means that $F = \bigcup_{i=1}^{n(m)} F_i$ is a small space. Therefore $R(f_F)$ may be much larger than the expected risk of our (true) target function f_0, since f_0 may not be in F. The distance between $R(f_F)$ and $R(f_0)$ is called the approximation error and can be bounded using results from approximation theory. We do not discuss these results here and refer the reader to Lorentz [1986] and DeVore [1998].

10.2.3 ϵ-uniform Convergence and the V_γ Dimension

As mentioned above finiteness of the VC-dimension is *not* a necessary condition for uniform convergence in the case of real valued functions. To get a necessary condition we need a slight extension of the VC-dimension that has been developed by Kearns and Schapire [1994] and Alon et al. [1997], known as the V_γ–dimension. Here we summarize the main results of that theory that we will also use later on to design regression machines for which we will have distribution independent uniform convergence. We begin with some definitions of the V_γ dimension, a variation of the fat-shattering dimension defined in Chapter 1. The V_γ dimension will the be measure of complexity used in this section.[9]

Definition 10.4
Let $A \leq c(\mathbf{x}, y, f(\mathbf{x})) \leq B$, $f \in F$, with A and $B < \infty$. The V_γ-dimension of c in F (of the set $\{c(\mathbf{x}, y, f(\mathbf{x})),\ f \in F\}$) is defined as the maximum number h of vectors $(\mathbf{x}_1, y_1) \ldots, (\mathbf{x}_h, y_h)$ that can be separated into two classes in all 2^h possible ways using rules:

$$\begin{aligned} &\text{class } 1 \quad \text{if} \quad c(\mathbf{x}, y_i, f(\mathbf{x}_i)) \geq s + \gamma \\ &\text{class } -1 \quad \text{if} \quad c(\mathbf{x}, y_i, f(\mathbf{x}_i)) \leq s - \gamma \end{aligned}$$

for $f \in F$ and some $s \in [\gamma + A, B - \gamma]$. If, for any number m, it is possible to find m points $(\mathbf{x}_1, y_1) \ldots, (\mathbf{x}_m, y_m)$ that can be separated in all the 2 possible ways, we will say that the V_γ-dimension of c in F is infinite.

Notice that if for each point $(\mathbf{x}_i, y_i)$ we use a different $s_i \geq 0$, we get the fat-shattering dimension (see Chapter 1). Furthermore, for $\gamma = 0$ this definition becomes the same as the definition for VC-dimension. Intuitively, for $\gamma > 0$ the "rule" for separating points is more restrictive than the rule in the case $\gamma = 0$. It requires that there is a "margin" between the points: points for which $c(\mathbf{x}, y, f(\mathbf{x}))$

8. Various cases are discussed in [Devroye et al., 1996], i.e., $n(m) = m$.
9. The fat-shattering dimension can also be used.

is between $s+\gamma$ and $s-\gamma$ are not classified. As a consequence, the V_γ dimension is a decreasing function of γ and in particular is smaller than the VC-dimension. If c is an indicator function, say $\theta(-yf(\mathbf{x}))$, then for any γ definition 10.4 reduces to that of the VC-dimension of a set of indicator functions. Generalizing slightly the definition of eq. (10.13) we will say that for a given $\epsilon > 0$ the ERM method converges ϵ-uniformly in F in probability, (or that there is ϵ-uniform convergence) if:

$$\lim_{m\to\infty} Pr\left\{\sup_{f\in F}|R_{\rm emp}(f)-R(f)|>\epsilon\right\}=0. \tag{10.26}$$

Notice that if eq. (10.26) holds for every $\epsilon > 0$ we have uniform convergence (eq. (10.13)). Moreover, it can be shown (variation of [Vapnik, 1998]) that ϵ-uniform convergence in probability implies that:

$$R(\hat{f}_m) \leq R(f_0) + 2\epsilon \tag{10.27}$$

in probability, where, as before, $\hat{f}_m$ is the minimizer of the empirical risk and f_0 is the minimizer of the expected expected risk in F.[10]

The basic theorems for the V_γ-dimension are the following:

Theorem 10.5 Alon et al., 1993
Let $A \leq c(\mathbf{x}, y, f(\mathbf{x}))) \leq B$, $f \in F$, F be a set of bounded functions. For any $\epsilon > 0$, if the V_γ dimension of c in F is finite for $\gamma = \beta\epsilon$ for some constant $\beta \geq \frac{1}{48}$, then the ERM method ϵ-converges in probability.

Theorem 10.6 Alon et al., 1993
Let $A \leq c(\mathbf{x}, y, f(\mathbf{x}))) \leq B$, $f \in F$, F be a set of bounded functions. The ERM method uniformly converges (in probability) if and only if the V_γ dimension of c in F is finite for every $\gamma > 0$. So finiteness of the V_γ dimension for every $\gamma > 0$ is a *necessary* **and** *sufficient* condition for distribution independent uniform convergence of the ERM method for real-valued functions.

Theorem 10.7 Alon et al., 1993
Let $A \leq c(\mathbf{x}, y, f(\mathbf{x})) \leq B$, $f \in F$, F be a set of bounded functions. For any $\epsilon \geq 0$, for all $m \geq \frac{2}{\epsilon^2}$ we have that if h_γ is the V_γ dimension of c in F for $\gamma = \beta\epsilon$ ($\beta \geq \frac{1}{48}$), h_γ finite, then:

$$Pr\left\{\sup_{f\in F}|R_{\rm emp}(f)-R(f)|>\epsilon\right\} \leq \mathcal{G}(\epsilon, m, h_\gamma), \tag{10.28}$$

where $\mathcal{G}$ is an increasing function of h_γ and a decreasing function of ϵ and m, with $\mathcal{G} \to 0$ as $m \to \infty$.[11]

10. This is like ϵ-learnability in the PAC model [Valiant, 1984].
11. Closed forms of $\mathcal{G}$ can be derived (see for example [Alon et al., 1997]) but we do not present them here for simplicity of notation.

From this theorem we can easily see that for any $\epsilon > 0$, for all $m \geq \frac{2}{\epsilon^2}$:

$$Pr\left\{R(\hat{f}_m) \leq R(f_0) + 2\epsilon\right\} \geq 1 - 2\mathcal{G}(\epsilon, m, h_\gamma), \tag{10.29}$$

where $\hat{f}_m$ is, as before, the minimizer of the empirical risk in F. An important observations to keep in mind is that theorem 10.7 requires the V_γ dimension of the loss function c in F. In the case of classification, this implies that if we want to derive bounds on the expected misclassification we have to use the V_γ dimension of the loss function $\theta(-yf(\mathbf{x}))$ (which is the $VC - dimension$ of the set of indicator functions $\{g(\mathbf{x}) := \text{sgn}\,(f(\mathbf{x})), f \in F\}$), and *not* the V_γ dimension of the set F.

The theory of the V_γ dimension justifies the "extended" SRM method we describe below. It is important to keep in mind that the method is only of theoretical interest and will only be used later as a theoretical motivation for RN and SVM.

Let m be the number of training data. For a fixed $\epsilon > 0$ such that $m \geq \frac{2}{\epsilon^2}$, let $\gamma = \frac{1}{48}\epsilon$, and consider, as before, a nested sequence of hypothesis spaces $F_1 \subset F_2 \subset \ldots \subset F_{n(m,\epsilon)}$, where each hypothesis space F_i has V_γ-dimension finite and larger than that of all previous sets, i.e., if h_i is the V_γ-dimension of space F_i, then $h_1 \leq h_2 \leq \ldots \leq h_{n(m,\epsilon)}$. For each element F_i of the structure consider the solution of the learning problem to be:

$$\hat{f}_{i,m} = \arg\min_{f \in F_i} R_{\text{emp}}(f). \tag{10.30}$$

Because of the way we define our structure the larger i is the smaller the empirical error of $\hat{f}_{i,m}$ is (since we have more "flexibility" to fit our training data), but the larger the right hand side of inequality (10.28) is. Using such a nested sequence of more and more complex hypothesis spaces, this *extended SRM* learning technique consists of finding the structure element $F_{n^*(m,\epsilon)}$ for which the trade off between empirical error and the right hand side of (10.28) is optimal. One practical idea is to find numerically for each F_i the "effective" ϵ_i so that the bound (10.28) is the same for all F_i, and then choose $\hat{f}_{i,m}$ for which the sum of $R_{\text{emp}}(f)$ and ϵ_i is minimized.

We *conjecture* that as $m \to \infty$, for appropriate choice of $n(m,\epsilon)$ with $n(m,\epsilon) \to \infty$ as $m \to \infty$, the expected risk of the solution of the method converges in probability to a value less than 2ϵ away from the minimum expected risk in $F = \bigcup_{i=1}^{\infty} F_i$. Notice that we described an SRM method for a fixed ϵ. If the V_γ dimension of F_i is finite for every $\gamma > 0$, we can further modify the extended SRM method so that $\epsilon \to 0$ as $m \to \infty$. We *conjecture* that if the (true) target function f_0 belongs to the closure of F, then as $m \to \infty$, with appropriate choices of ϵ, $n(m,\epsilon)$ and $n^*(m,\epsilon)$ the solution of this SRM method can be proven (as before) to satisfy eq. (10.11) in probability. Finding appropriate forms of ϵ, $n(m,\epsilon)$ and $n^*(m,\epsilon)$ is an open theoretical problem (which we believe to be a technical matter). Again, as in the case of "standard" SRM, in practice m is finite so $F = \bigcup_{i=1}^{n(m,\epsilon)} F_i$ is a small space and the solution of this method may have expected risk much larger that the expected risk of the (true) target function. Approximation theory can be used to bound this difference [Niyogi and Girosi, 1996].

The proposed method is difficult to implement in practice since it is difficult to decide the optimal trade off between empirical error and the bound (10.28). If we had constructive bounds on the deviation between the empirical and the expected risk like that of equation (10.24) then we could have a practical way of choosing the optimal element of the structure. Unfortunately existing bounds of that type ([Alon et al., 1997, Bartlett et al., 1996], also Section 1.2) are not tight. So the final choice of the element of the structure may be done in practice using other techniques such as cross-validation [Wahba, 1990].

10.2.4 Overview of our Approach

In order to set the stage for the next two sections on regularization and Support Vector Machines, we outline here how we can justify the proper use of the RN and the SVM functionals (see (10.3)) in the framework of the SRM principles just described.

The basic idea is to define a structure in terms of a nested sequence of hypothesis spaces $F_1 \subset F_2 \subset \ldots \subset F_{n(m)}$ with F_r being the set of functions f in the RKHS with:

$$\|f\|_k^2 \leq A_r, \tag{10.31}$$

where A_r is a monotonically increasing sequence of positive constants. Following the SRM method outlined above, for each m we will minimize the empirical risk

$$\frac{1}{m}\sum_{i=1} c(\mathbf{x}_i, y_i, f(\mathbf{x})), \tag{10.32}$$

subject to the constraint (10.31). This in turn leads to using the Lagrange multiplier λ_r and to minimizing

$$\frac{1}{m}\sum_{i=1} c(\mathbf{x}_i, y_i, f(\mathbf{x})) + \lambda_r(\|f\|_k^2 - A_r), \tag{10.33}$$

with respect to f and maximizing with respect to $\lambda_r \geq 0$ for each element of the structure. We can then choose the optimal $n^*(m)$ and the associated $\lambda^*(m)$, and get the optimal solution $\hat{f}_{n^*(m)}$.

The solution we get using this method is clearly the same as the solution of:

$$\frac{1}{m}\sum_{i=1} c(\mathbf{x}_i, y_i, f(\mathbf{x})) + \lambda^*(m)\|f\|_k^2 \tag{10.34}$$

where $\lambda^*(m)$ is the optimal Lagrange multiplier corresponding to the optimal element of the structure $A_{n^*(m)}$.

Notice that this approach is quite general. In particular it can be applied to standard L_2 regularization, to SVM regression, and, as we will see, to SVM classification with the appropriate $c(\cdot,\cdot,\cdot)$.

In Section 10.5 we will describe this approach in more detail. We have outlined this theoretical method here so that the reader understands our motivation for

reviewing in the next two sections the approximation schemes resulting from the minimization of functionals of the form of equation (10.34) for the three loss function (10.4), (10.5) and (10.6).

10.3 Regularization Networks

In this section we consider the approximation scheme that arises from the minimization of the quadratic functional

$$\min_{f \in F} R_{\text{reg}}[f] = \frac{1}{m} \sum_{i=1} (y_i - f(\mathbf{x}_i))^2 + \lambda \|f\|_k^2 \tag{10.35}$$

for a fixed λ. Formulations like equation (10.35) are a special form of regularization theory developed by Tikhonov and Arsenin [1977], Ivanov [1976] and others to solve ill-posed problems and in particular to solve the problem of approximating the functional relation between $\mathbf{x}$ and y given a finite number of examples $X \times Y = \{\mathbf{x}_i, y_i\}_{i=1}$. As we mentioned in the previous sections our motivation in this chapter is to use this formulation as an approximate implementation of Vapnik's SRM principle.

In standard regularization the data term is an L_2 loss function for the empirical risk, whereas the second term – called *stabilizer* – is usually written as a functional $\Omega(f)$ with certain properties [Tikhonov and Arsenin, 1977, Poggio and Girosi, 1989, Girosi et al., 1995]. Here we consider a special class of stabilizers, that is the norm $\|f\|_k^2$ in a RKHS induced by a symmetric, positive definite function $k(\mathbf{x}, \mathbf{y})$. This choice allows us to develop a framework of regularization which includes most of the usual regularization schemes. The only significant omission in this treatment – that we make here for simplicity – is the restriction on k to be symmetric positive definite so that the stabilizer is a norm. However, the theory can be extended without problems to the case in which k is positive semidefinite, in which case the stabilizer is a semi-norm [Wahba, 1990, Madych and Nelson, 1990a, Dyn, 1991, Dyn et al., 1986]. This approach was also sketched in [Smola and Schölkopf, 1998b].

The stabilizer in equation (10.35) effectively constrains f to be in the RKHS defined by the positive definite kernel k. It is possible to show (see for example [Poggio and Girosi, 1989, Girosi et al., 1995]) that the function that minimizes the functional (10.35) has the form:

$$f(\mathbf{x}) = \sum_{i=1} \alpha_i k(\mathbf{x}, \mathbf{x}_i), \tag{10.36}$$

the coefficients α_i depend on the data and satisfy the following linear system of equations:

$$(K + \lambda I)\boldsymbol{\alpha} = \mathbf{y} \tag{10.37}$$

where I is the identity matrix, and we have defined

$$(\mathbf{y})_i = y_i \ , \quad (\boldsymbol{\alpha})_i = \alpha_i \ , \quad (K)_{ij} = k(\mathbf{x}_i, \mathbf{x}_j). \tag{10.38}$$

It is remarkable that the solution of the more general case of

$$\min_{f \in F} R_{\text{reg}}[f] = \frac{1}{m} \sum_{i=1} c(\mathbf{x}_i, y_i, f(\mathbf{x})) + \lambda \|f\|_k^2, \tag{10.39}$$

where the function c is any differentiable function, is quite similar: the solution has exactly the same general form of (10.36), though the coefficients cannot be found anymore by solving a linear system of equations as in equation (10.37) [Girosi, 1991, Girosi et al., 1991, Smola and Schölkopf, 1998b].

The approximation scheme of equation (10.36) has a simple interpretation in terms of a network with one layer of hidden units [Poggio and Girosi, 1992, Girosi et al., 1995]. Using different kernels we get various RN's. A short list of examples is given in Table 10.1.

Kernel Function	Regularization Network
$k(\mathbf{x}-\mathbf{y}) = \exp(-\|\mathbf{x}-\mathbf{y}\|^2)$	Gaussian RBF
$k(\mathbf{x}-\mathbf{y}) = (\|\mathbf{x}-\mathbf{y}\|^2 + c^2)^{-\frac{1}{2}}$	Inverse Multiquadric
$k(\mathbf{x}-\mathbf{y}) = (\|\mathbf{x}-\mathbf{y}\|^2 + c^2)^{\frac{1}{2}}$	Multiquadric
$k(\mathbf{x}-\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|^{2n+1}$ $k(\mathbf{x}-\mathbf{y}) = \|\mathbf{x}-\mathbf{y}\|^{2n} \ln(\|\mathbf{x}-\mathbf{y}\|)$	Thin plate splines
$k(\mathbf{x},\mathbf{y}) = \tanh(\mathbf{x}\cdot\mathbf{y}-\theta)$	(only for some values of θ) Multi Layer Perceptron
$k(\mathbf{x},\mathbf{y}) = (1+\mathbf{x}\cdot\mathbf{y})^d$	Polynomial of degree d
$k(x,y) = B_{2n+1}(x-y)$	B-splines
$k(x,y) = \frac{\sin(d+1/2)(x-y)}{\sin\frac{(x-y)}{2}}$	Trigonometric polynomial of degree d

Table 10.1 Some possible kernel functions. The first four are radial kernels. The multiquadric and thin plate splines are positive semidefinite and thus require an extension of the simple RKHS theory of this chapter. The last three kernels are listed in [Vapnik, 1998]. Polynomial Kernel were used in [Poggio, 1975], and B-spline in [Vapnik et al., 1997]. The last two kernels are one-dimensional: multidimensional kernels can be built by tensor products of one-dimensional ones. The functions B_n are piecewise polynomials of degree n, whose exact definition can be found in [Schumaker, 1981].

When the kernel k is positive semidefinite, there is a subspace of functions f which have norm $\|f\|_k^2$ equal to zero. They form the null space of the functional $\|f\|_k^2$ and in this case the minimizer of (10.35) has the form [Wahba, 1990]:

$$f(\mathbf{x}) = \sum_{i=1} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + \sum_{j=1}^{l} b_j \phi_j(\mathbf{x}), \tag{10.40}$$

where $\{\phi_j\}_{j=1}^{l}$ is a basis in the null space of the stabilizer, which in most cases is a set of polynomials, and therefore will be referred to as the "polynomial term" in equation (10.40). The coefficients b_j and α_i depend on the data. For the standard regularization case of equation (10.35), the coefficients of equation (10.40) satisfy the following linear system:

$$(K + \lambda I)\boldsymbol{\alpha} + \Phi^T \mathbf{b} = \mathbf{y}, \tag{10.41}$$

$$\Phi\boldsymbol{\alpha} = 0, \tag{10.42}$$

where I is the identity matrix, and we have defined

$$(\mathbf{y})_i = y_i \ , \quad (\boldsymbol{\alpha})_i = \alpha_i \ , \quad (\mathbf{b})_i = b_i \ , \tag{10.43}$$

$$(K)_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) \ , \quad (\Phi)_{ji} = \phi_j(\mathbf{x}_i). \tag{10.44}$$

When the kernel is positive definite, as in the case of the Gaussian, the null space of the stabilizer is empty. However, it is often convenient to redefine the kernel and the norm induced by it so that the induced RKHS contains only zero-mean functions, that is functions $f_1(\mathbf{x})$ s.t. $\int_X f_1(\mathbf{x})dx = 0$. In the case of a radial kernel k, for instance, this amounts to considering a new kernel

$$k'(\mathbf{x}, \mathbf{y}) = k(\mathbf{x}, \mathbf{y}) - \lambda_0 \tag{10.45}$$

without the zeroth order Fourier component, and a norm

$$\|f\|_{k'}^2 = \|f\|_k^2 - \frac{a_0^2}{\lambda_0} \tag{10.46}$$

where λ_0 is the eigenvalue corresponding to the zeroth order Fourier component, and a_0 is the coefficient of f corresponding to that component. The null space induced by the new k' is the space of constant functions. Then the minimizer of the corresponding functional (10.35) has the form:

$$f(\mathbf{x}) = \sum_{i=1} \alpha_i k'(\mathbf{x}, \mathbf{x}_i) + b, \tag{10.47}$$

with the coefficients satisfying equations (10.41) and (10.42), that respectively become:

$$(K' + \lambda I)\boldsymbol{\alpha} + \mathbf{1}b = (K - \lambda_0 I + \lambda I)\boldsymbol{\alpha} + \mathbf{1}b = (K + (\lambda - \lambda_0)I)\boldsymbol{\alpha} + \mathbf{1}b = \mathbf{y}, \tag{10.48}$$

$$\sum_{i=1} \alpha_i = 0. \tag{10.49}$$

Equations (10.47) and (10.49) imply that the the minimizer of (10.35) is of the form:

$$f(\mathbf{x}) = \sum_{i=1} \alpha_i k'(\mathbf{x}, \mathbf{x}_i) + b = \sum_{i=1} \alpha_i (k(\mathbf{x}, \mathbf{x}_i) - \lambda_0) + b = \sum_{i=1} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b. \tag{10.50}$$

Thus we can effectively use a positive definite k *and* the constant b, since the only change in equation (10.48) just amounts to the use of a different λ. Choosing to use a non-zero b effectively means choosing a different feature space and a

different stabilizer from the usual case of equation (10.35): the constant feature is not considered in the RKHS norm and therefore is not "penalized." This choice is often quite reasonable, since in many regression and, especially, classification problems, shifts by a constant in f should not be penalized.

In summary, the argument of this section shows that using a RN of the form (10.50) (for a certain class of kernels k) is equivalent to minimizing functionals such as (10.35) or (10.39). The choice of k is equivalent to the choice of a corresponding RKHS and leads to various standard learning techniques such as Radial Basis Function networks.

Notice that in the framework we use here the kernels k are not required to be radial or even shift-invariant. Regularization techniques used to solve supervised learning problems [Poggio and Girosi, 1989, Girosi et al., 1995] were typically used with shift invariant stabilizers (tensor product and additive stabilizers are exceptions, see [Girosi et al., 1995]).

10.3.1 From Regression to Classification

So far we only considered the case that the unknown function can take any real values, specifically the case of regression. In the particular case that the unknown function takes only two values, i.e., -1 and 1, we have the problem of binary pattern classification, i.e., the case where we are given data that belong to one of two classes (classes -1 and 1) and we want to find a function that separates these classes. It can be shown [Duda and Hart, 1973] that, if c in equation (10.39) is $(y - f(\mathbf{x}))^2$, and if k defines a finite dimensional RKHS, then the minimizer of the equation

$$R_{\text{reg}}[f] = \frac{1}{m}\sum_{i=1}(f(\mathbf{x}_i) - y_i)^2 + \lambda\|f\|_k^2, \tag{10.51}$$

for $\lambda \to 0$ approaches *asymptotically* the function in the RKHS that is closest in the L_2 norm to the regression function:

$$f_0(\mathbf{x}) = P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x}) \tag{10.52}$$

The *optimal Bayes rule classifier* is given by thresholding the regression function, i.e., by $\text{sign}(f_0(\mathbf{x}))$. Notice that in the case of infinite dimensional RKHS asymptotic results ensuring consistency are available (see [Devroye et al., 1996, Theorem 29.8]) but depend on several conditions that are not automatically satisfied in the case we are considering. The Bayes classifier is the best classifier, given the correct probability distribution P. However, approximating function (10.52) in the RKHS in L_2 does not necessarily imply that we find the best approximation to the Bayes classifier. For classification, only the sign of the regression function matters and not the exact value of it. Notice that an approximation of the regression function using a mean square error criterion places more emphasis on the most probable data points and not on the most "important" ones which are the ones near the separating boundary.

In the next section we will study Vapnik's more natural approach to the problem of classification that is based on choosing a loss function c different from the square error. This approach leads to solutions that emphasize data points near the separating surface.

10.4 Support Vector Machines

In this section we discuss the technique of Support Vector Machines (SVM) for Regression (SVMR) [Vapnik, 1995, 1998] in terms of the SVM functional. We then show the SVM for binary pattern classification can be derived as a special case of the regression formulation [Pontil et al., 1998b].

10.4.1 SVM in RKHS

Once again the problem is to learn a functional relation between $\mathbf{x}$ and y given a finite number of examples $X \times Y$.

The method of SVMR corresponds to the following functional

$$R_{\text{reg}}[f] = \frac{1}{m}\sum_{i=1} |y_i - f(\mathbf{x}_i)|_\epsilon + \lambda \|f\|_k^2 \tag{10.53}$$

which is a special case of equation (10.39) and where

$$|x|_\epsilon \equiv \begin{cases} 0 & \text{if } |x| < \epsilon \\ |x| - \epsilon & \text{otherwise,} \end{cases} \tag{10.54}$$

is the $\epsilon-$Insensitive Loss Function (ILF) (also noted with L_ϵ). Note that the ILF assigns zero cost to errors smaller then ϵ. In other words, for the cost function $|\cdot|_\epsilon$ any function closer than ϵ to the data points is a perfect interpolant. We can think of the parameter ϵ as the resolution at which we want to look the data. For this reason we expect that the larger ϵ is, the simpler the representation will be. We will come back to this point in Section 10.7.

The minimizer of R_{reg} in the RKHS $\mathcal{H}_k$ defined by the kernel k has the general form given by equation (10.50), that is

$$f(\mathbf{x}) = \sum_{i=1} \alpha_i k(\mathbf{x}_i, \mathbf{x}) + b, \tag{10.55}$$

where we can include the constant b for the same reasons discussed in Section 10.3. The coefficient α_i are found by solving the following problem:

Problem 10.8

$$\min_\alpha \mathcal{R}[\boldsymbol{\alpha}] = \frac{1}{2}\sum_{i,j=1} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1} \alpha_i y_i + \epsilon \sum_{i=1} |\alpha_i| \tag{10.56}$$

subject to the constraints

$$\sum_{i=1} \alpha_i = 0 \text{ and } -\frac{C}{m} \leq \alpha_i \leq \frac{C}{m} \text{ for all } i = 1, \ldots, m. \tag{10.57}$$

The support vectors (SV) are the points for which $|\alpha_i| > 0$. Points at which the error is smaller than ϵ are never support vectors, and do not enter in the determination of the solution. A consequence of this fact is that if the SVM were run again on the new data set consisting of only the SVs the same solution would be found. Finally notice that by setting $\alpha_i = \bar{\alpha}_i + \bar{\alpha}_i^*$, with $\bar{\alpha}_i, \bar{\alpha}_i^* \geq 0$ we find the standard quadratic programming formulation of SVMR [Vapnik, 1998].

10.4.2 From Regression to Classification

In the previous section we discussed the connection between regression and classification in the framework of regularization. In this section, after stating the formulation of SVM for binary pattern classification (SVMC) as developed by Cortes and Vapnik [1995], we discuss a connection between SVMC and SVMR. We will not discuss the theory of SVMC here; we refer the reader to [Vapnik, 1998]. We point out that the SVM technique has first been proposed for binary pattern classification problems and then extended to the general regression problem by Vapnik [1995]. Here our primary focus is regression and we consider classification as a special case of regression. SVMC can be formulated as the problem of minimizing:

$$R_{\text{reg}}[f] = \frac{1}{m} \sum_i \theta(1 - y_i f(\mathbf{x}_i))(1 - y_i f(\mathbf{x}_i)) + \frac{1}{2C} \|f\|_k^2, \tag{10.58}$$

which is again of the form (10.3). Using the fact that $y_i \in \{-1, +1\}$ it is easy to see that our formulation (equation (10.58)) is equivalent to the following quadratic programming problem, originally proposed by Cortes and Vapnik [1995]:

Problem 10.9

$$\min_{f \in \mathcal{H}_k, \boldsymbol{\xi}} \Phi(f, \boldsymbol{\xi}) = \frac{C}{m} \sum_{i=1} \xi_i + \frac{1}{2} \|f\|_k^2 \tag{10.59}$$

subject to the constraints:

$$\begin{aligned} y_i f(\mathbf{x}_i) &\geq 1 - \xi_i, \quad \text{for all } i = 1, \ldots, m \\ \xi_i &\geq 0, \quad \text{for all } i = 1, \ldots, m. \end{aligned} \tag{10.60}$$

The solution of this problem is again of the form (10.55), where it turns out that $0 \leq \alpha_i \leq \frac{C}{m}$. The input data points $\mathbf{x}_i$ for which α_i is different from zero are called, as in the case of regression, *support vectors* (SVs). It is often possible to write the solution $f(\mathbf{x})$ as a linear combination of SVs in a number of different ways (for example in case that the feature space induced by the kernel k has dimensionality lower than the number of SVs). The SVs that appear in *all* these linear combinations are called *essential support vectors*.

Roughly speaking the motivation for problem (10.9) is to minimize the empirical error measured by $\sum_{i=1} \xi_i$[12] while controlling capacity measured in terms of the norm of f in the RKHS. In fact, the norm of f is related to the notion of *margin*, an important idea for SVMC for which we refer the reader to [Vapnik, 1998, Burges, 1998].

We now address the following question: what happens if we apply the SVMR formulation (10.53) to the binary pattern classification case, i.e., the case where y_i take values $\{-1, 1\}$, treating classification as a regression on binary data?

It is possible to show that for a given constant C in problem (10.9), there exist C and ϵ in problem (10.8) such that the solutions of the two problems are the same, up to a constant factor. This is summarized in the following theorem:

Theorem 10.10
Suppose the classification problem (10.9) is solved with parameter C, and the optimal solution is found to be f. Then, there exists a value $a \in (0, 1)$ such that for $\forall \epsilon \in [a, 1)$, if problem (10.8) is solved with parameter $(1-\epsilon)C$, the optimal solution will be $(1-\epsilon)f$.

We refer to [Pontil et al., 1998b] for the proof. A direct implication of this result is that one can solve any SVMC problem through the SVMR formulation. It is an open question what theoretical implications Theorem 10.10 may have about SVMC and SVMR. In particular in Section 10.5 we will discuss some recent theoretical results on SVMC that have not yet been extended to SVMR. It is possible that Theorem 10.10 may help to extend them to SVMR.

10.5 SRM for RNs and SVMs

At the end of Section 10.2 we outlined how one should implement both RN and SVM according to SRM. To use the standard SRM method we first need to know the VC-dimension of the hypothesis spaces we use. In Sections 10.3 and 10.4 we saw that both RN and SVM use as hypothesis spaces sets of bounded functions f in a RKHS with $||f||_k^2$ bounded (i.e., $||f||_k^2 \leq A$), where k is the kernel of the RKHS. So in order to use the standard SRM method outlined in Section 10.2 we need to know the VC dimension of such spaces under the loss functions of RN and SVM.

Unfortunately it can be shown that when the loss function c is $(y - f(\mathbf{x}))^2$ (the L_2) and also when it is $|y_i - f(\mathbf{x}_i)|_\epsilon$ (the L_ϵ), the VC-dimension of $c(\mathbf{x}_i, y, f(\mathbf{x}))$ with f in $F_A = \{f : ||f||_k^2 \leq A\}$ does not depend on A, and is infinite if the RKHS

12. As we mentioned in Section 10.2, for binary pattern classification the empirical error is defined as a sum of binary numbers which in problem (10.9) would correspond to $\sum_{i=1} \theta(\xi_i)$. However in such a case the minimization problem becomes computationally intractable. This is why in practice in the cost functional $\Phi(f, \boldsymbol{\xi})$ we approximate $\theta(\xi_i)$ with ξ_i. We discuss this further in Section 10.5.

is infinite dimensional. More precisely we have the following theorem (for a proof see for example [Williamson et al., 1998, Evgeniou and Pontil, 1999b])

Theorem 10.11
Let M be the dimensionality of a RKHS $\mathcal{H}_k$. For both the L_2 and the ϵ-insensitive loss function c, the VC-dimension of c in the space $F_A = \{f \in \mathcal{H}_k : ||f||_k^2 \leq A\}$ is $O(M)$, *independently* of A. Moreover, if M is infinite, the VC-dimension is infinite for any A.

It is thus impossible to use SRM with this kind of hypothesis spaces: in the case of finite dimensional RKHS, the RKHS norm of f cannot be used to define a structure of spaces with different VC-dimensions, and in the (typical) case that the dimensionality of the RKHS is infinite, it is not even possible to use bound (10.24). So *the VC-dimension cannot be used directly neither for RN nor for SVMR.*

On the other hand, we can still use the V_γ dimension and the extended SRM method outlined in Section 10.2. Again we need to know the V_γ dimension of our loss function c in the space F_A defined above. In the typical case that the input space $\mathcal{X}$ is bounded, the V_γ dimension does depend on A and is not infinite in the case of infinite dimensional RKHS. More precisely the following theorem holds (for a proof see [Evgeniou and Pontil, 1999b]):

Theorem 10.12
Let M be the dimensionality of a RKHS $\mathcal{H}_k$ with kernel k. Assume our input space $\mathcal{X}$ is bounded and let R be the radius of the smallest ball B containing the data $\mathbf{x}$ in the feature space induced by kernel k. The V_γ dimension h for regression using L_2 or L_ϵ loss functions for hypothesis spaces $F_A = \{f \in \mathcal{H}_k \mid \|f\|_k \leq A\}$ and y bounded, is finite for $\forall\ \gamma > 0$, with $h \leq \mathrm{O}(\min\ (M, \frac{(R^2+1)(A^2+1)}{\gamma^2}))$.

Notice that for fixed γ and fixed radius of the data the only variable that controls the V_γ dimension is the upper bound on the RKHS norm of the functions, namely A. Moreover, the V_γ dimension is finite for $\forall\ \gamma > 0$, therefore, according to Theorem 10.6, ERM uniformly converges in F_A for any $A < \infty$, both for RN and for SVMR. Thus both RNs and SVMR are *consistent* in F_A for any $A < \infty$. So, theoretically, we can use the extended SRM method with a sequence of hypothesis spaces F_A each defined for different As. To repeat, for a fixed $\gamma > 0$ (we can let γ go to 0 as $m \to \infty$) we first define a structure $F_1 \subset F_2 \subset \ldots \subset F_{n(m)}$ where F_r is the set of bounded functions f in a RKHS with $||f||_k^2 \leq A_r$, $A_r < \infty$, and the numbers A_r form an increasing sequence. Then we minimize the empirical risk in each F_r by solving the problem:

$$\text{minimize } \frac{1}{m}\sum_{i=1} c(\mathbf{x}_i, y_i, f(\mathbf{x}_i)) \text{ subject to } ||f||_k^2 \leq A_r \tag{10.61}$$

To solve this minimization problem we minimize

$$\frac{1}{m}\sum_{i=1} c(\mathbf{x}_i, y_i, f(\mathbf{x}_i)) + \lambda_r(||f||_k^2 - A_r) \tag{10.62}$$

with respect to f and maximize with respect to the Lagrange multiplier λ_r. If f_r is the solution of this problem, at the end we choose the optimal $f_{n^*(m)}$ in $F_{n^*(m)}$ with the associated $\lambda_{n^*(m)}$, where optimality is decided based on a trade off between empirical error and the bound (10.28) for the fixed γ (which, as we mentioned, can approach zero). In the case of RN, c is the L_2 loss function, whereas in the case of SVMR it is the ϵ-insensitive loss function.

In practice it is difficult to implement the extended SRM for two main reasons. First, as we discussed in Section 10.2, SRM using the V_γ dimension is practically difficult because we do not have tight bounds to use in order to pick the optimal $F_{n^*(m)}$ (combining theorems 10.12 and 10.7 bounds on the expected risk of RN and SVMR machines of the form (10.61) can be derived - see also a similar bound in Chapter 1 - but these bounds are not practically useful). Second, even if we could make a choice of $F_{n^*(m)}$, it is computationally difficult to implement the SRM since problem (10.61) is a constrained minimization one with non-linear constraints, and solving such a problem for a number of spaces F_r can be computationally difficult. So implementing SRM using the V_γ dimension of nested subspaces of a RKHS is practically a very difficult problem.

On the other hand, if we had the optimal Lagrange multiplier $\lambda_{n^*(m)}$, we could simply solve the unconstrained minimization problem:

$$\frac{1}{m}\sum_{i=1} c(\mathbf{x}_i, y_i, f(\mathbf{x}_i)) + \lambda_{n^*(m)}||f||_k^2 \tag{10.63}$$

both for RN and for SVMR. This is exactly the problem we solve in practice, as we described in Sections 10.3 and 10.4. Since the value $\lambda_{n^*(m)}$ is not known in practice, we can only "implement" the extended SRM approximately by minimizing (10.63) with various values of λ and then picking the best λ using techniques such as cross-validation [Allen, 1974, Wahba, 1980, 1985, Kearns et al., 1997], Generalized Cross Validation, Finite Prediction Error and the MDL criteria (see [Vapnik, 1998] for a review and comparison). It is important to notice that bound (10.28) does not hold if we use the norm of the solution of (10.63) instead of A in Theorem 10.12. We discuss this issue below.

Summarizing, both the RN and the SVMR methods discussed in Sections 10.3 and 10.4 can be seen as approximations of the extended SRM method using the V_γ dimension, with nested hypothesis spaces being of the form $F_A = \{f \in \mathcal{H}_k : ||f||_k^2 \leq A\}$, $\mathcal{H}_k$ being a RKHS defined by kernel k. For both RN and SVMR the V_γ dimension of the loss function c in F_A is finite for $\forall\ \gamma > 0$, so the ERM method uniformly converges in F_A for any $A < \infty$, and we can use the extended SRM method outlined in Section 10.2.

10.5.1 SRM for SVM Classification

It is interesting to notice that the same analysis can be used for the problem of classification. In this case the following theorem holds [Evgeniou and Pontil, 1999a]:

Theorem 10.13
Let M be the dimensionality of a RKHS $\mathcal{H}_k$ with kernel k. Assume our input space $\mathcal{X}$ is bounded and let R be the radius of the sphere where our data $\mathbf{x}$ belong to, in the feature space induced by kernel k. The V_γ dimension of the soft margin loss function $\theta(1 - yf(\mathbf{x}))(1 - yf(\mathbf{x}))$ in $F_A = \{f \in \mathcal{H}_k : ||f||_k \leq A\}$ is $\leq O(\min(M, \frac{R^2A^2}{\gamma^2}))$. In the case that M is infinite the V_γ dimension becomes $\leq O(\frac{R^2A^2}{\gamma^2})$, which means it is finite for $\forall\ \gamma > 0$.

This theorem, combined with the theorems on V_γ dimension summarized in Section 10.2, can be used for a distribution *independent* analysis of SVMC (of the form (10.61)) like that of SVMR and RN. However, a direct application of theorems 10.13 and 10.7 leads to a bound on the expected soft margin error of the SVMC solution, instead of a more interesting bound on the expected *misclassification* error. We can bound the expected *misclassification* error as follows.

Using Theorem 10.7 with the soft margin loss function we can get a bound on the expected soft margin loss in terms of the empirical one (the $\sum_{i=1} \xi_i$ of problem 10.9) and the V_γ dimension given by theorem 10.13. Theorem 10.7 implies:

$$Pr\Big\{ \sup_{f \in F_A} |R_{\text{emp}}(f) - R(f)| > \epsilon \Big\} \leq \mathcal{G}(\epsilon, m, h_\gamma), \tag{10.64}$$

where both the expected and the empirical errors are measured using the soft margin loss function, and h_γ is the V_γ dimension of Theorem 10.13 for $\gamma = \alpha\epsilon$ and α as in Theorem 10.7. On the other hand, $\theta(-yf(\mathbf{x})) \leq \theta(1 - yf(\mathbf{x}))(1 - yf(\mathbf{x}))$ for $\forall\ (\mathbf{x}, y)$, which implies that the expected misclassification error is less than the expected soft margin error. Using (10.64) we get that (uniformly) for all $f \in F_A$:

$$Pr\{R(f) > \epsilon + R_{\text{emp}}(f)\} \leq \mathcal{G}(\epsilon, m, h_\gamma), \tag{10.65}$$

Notice that (10.65) is different from existing bounds that use the empirical hard margin ($\theta(1 - yf(\mathbf{x}))$) error (see Chapter 1 or [Bartlett and Shawe-Taylor, 1999]). It is similar in spirit to bounds in Chapter 19 where the $\sum_{i=1} \xi_i^2$ is used.[13] On the other hand, it can be shown [Evgeniou and Pontil, 1999a] that the V_γ dimension for loss functions of the form $\theta(1 - yf(\mathbf{x}))(1 - yf(\mathbf{x}))^\sigma$ is of the form $O(\frac{R^2A^2}{\gamma^{\frac{2}{\sigma}}})$ for $\forall\ 0 < \sigma \leq 1$. Thus, using the same approach outlined above for the soft margin, we can get bounds on the misclassification error of SVMC in terms of $\sum_{i=1}(\xi_i)^\sigma$, which, for σ near 0, is close to the margin error used for the bounds in Chapter 1 and in [Bartlett and Shawe-Taylor, 1999]. It is important to point out that bounds like (10.65) hold only for the machines of the form (10.61), and not for the machines of the form (10.3) typically used in practice (Evgeniou and Pontil [1999a]). This is unlike the bound in Bartlett and Shawe-Taylor [1999] which holds for machines of the form (10.61) and is derived using the theoretical results of Bartlett [1998] where a type of "continuous" SRM (for example for a structure of hypothesis spaces

13. The $\sum_{i=1} \xi_i$ can be very different from the hard margin (or the misclassification) error. This may lead to various pathological situations (cf., e.g., [Rifkin et al., 1999]).

defined through the continuous parameter A of (10.61)) is studied.[14] For more information we refer the reader to Bartlett [1998], Evgeniou and Pontil [1999a].

In the case of classification the difficulty is the minimization of the empirical misclassification error. Notice that SVMC does *not* minimize the misclassification error, and instead minimizes the empirical error using the soft margin loss function. One can still use the SRM method with the soft margin loss function (10.6), in which case minimizing the empirical risk is possible. The SRM method with the soft margin loss function would be consistent, but the misclassification error of the solution may not be minimal. It is unclear whether SVMC is consistent in terms of misclassification error. In fact the V_γ dimension of the misclassification loss function (which is the same as the VC-dimension - see Section 10.2) is known to be equal to the dimensionality of the RKHS plus one [Vapnik, 1998]. This implies that, as discussed at the beginning of this section, it cannot be used to study the expected misclassification error of SVMC in terms of the empirical one.

10.5.1.1 Distribution Dependent Bounds for SVMC

We close this section with a brief reference to a recent distribution *dependent* result on the generalization error of SVMC. This result does not use the V_γ or VC dimensions, which, as we mentioned in Section 10.2, are used only for distribution *independent* analysis. It also leads to bounds on the performance of SVMC that (unlike the distribution independent ones) can be useful in practice.[15]

For a given training set of size m, let us define SV_m to be the number of essential support vectors of SVMC, (as we defined them in Section 10.4). Let R_m be the radius of the smallest hypersphere in the feature space induced by kernel k containing all essential SVs, $\|f\|_k^2(m)$ the norm of the solution of SVMC, and $\rho(m) = \frac{1}{\|f\|_k^2(m)}$ the margin. Then for a fixed kernel and for a fixed value of the SVMC parameter C the following theorem holds:

Theorem 10.14 Vapnik [1998]
The expected misclassification risk of the SVM trained on m data points sampled from $\mathbb{R}^N \times \mathbb{R}$ according to a probability distribution $p(\mathbf{x}, y)$ is bounded by:

$$E\left\{\frac{\min\left(SV_{m+1}, \ \frac{R_{m+1}^2}{\rho(m+1)}\right)}{m+1}\right\} \tag{10.66}$$

where the expectation E is taken over $p(\mathbf{x}, y)$.

This theorem can also be used to justify the current formulation of SVMC, since minimizing $||f||_k^2(m)$ (which is what we do in SVMC) affects the bound of Theorem 10.14. It is an open question whether the bound of (10.14) can be used to

14. It is important to notice that all these bounds are not tight enough in practice.
15. Further distribution dependent results have been derived recently - see Chapter 19.

construct learning machines that are better than current SVM. The theorem suggests that a learning machine should, instead of only minimizing $||f||_k^2$, minimize $\min\left(SV_m, \ \frac{R_{m+1}^2}{\rho(m+1)}\right)$. Finally, it is an open question whether similar results exist for the case of SVMR. As we mentioned in Section 10.4, the connection between SVMC and SVMR outlined in that section may suggest how to extend such results to SVMR. The problem of finding better distribution dependent results on the generalization capabilities of SVM is a topic of current research which may lead to better learning machines.

10.6 A Bayesian Interpretation of Regularization and SRM?

10.6.1 Maximum A Posteriori Interpretation of Regularization

It is well known that a variational principle of the type of equation (10.1) can be derived not only in the context of functional analysis [Tikhonov and Arsenin, 1977], but also in a probabilistic framework [Kimeldorf and Wahba, 1971, Wahba, 1990, 1980, Poggio et al., 1985, Marroquin et al., 1987, Bertero et al., 1988]. In this section we illustrate this connection for both RN and SVM, in the setting of RKHS. Consider the standard regularization case

$$\min_{f \in \mathcal{H}_k} R_{\text{reg}}[f] = \frac{1}{m} \sum_{i=1} (y_i - f(\mathbf{x}_i))^2 + \lambda \|f\|_k^2 \tag{10.67}$$

Following Girosi et al. [1995] let us define:

1. $X \times Y = \{(\mathbf{x}_i, y_i)\}$ for $i = 1, \cdots, m$ to be the set of training examples, as in the previous sections.

2. $\mathcal{P}[f|X \times Y]$ as the conditional probability of the function f given the examples $X \times Y$.

3. $\mathcal{P}[X \times Y|f]$ as the conditional probability of $X \times Y$ given f. If the function underlying the data is f, this is the probability that by random sampling the function f at the sites $\{\mathbf{x}_i\}_{i=1}$ the set of measurement $\{y_i\}_{i=1}$ is obtained. This is therefore a model of the noise.

4. $\mathcal{P}[f]$: is the *a priori* probability of the random field f. This embodies our *a priori* knowledge of the function, and can be used to impose constraints on the model, assigning significant probability only to those functions that satisfy those constraints.

Assuming that the probability distributions $\mathcal{P}[X \times Y|f]$ and $\mathcal{P}[f]$ are known, the posterior distribution $\mathcal{P}[f|X \times Y]$ can now be computed by applying the Bayes rule:

$$\mathcal{P}[f|X \times Y] \propto \mathcal{P}[X \times Y|f] \ \mathcal{P}[f]. \tag{10.68}$$

If the noise is normally distributed with variance σ, then the probability $\mathcal{P}[X \times Y|f]$ can be written as:

$$\mathcal{P}[X \times Y|f] \propto e^{-\frac{1}{2\sigma^2}\sum_{i=1}(y_i-(\mathbf{x}_i))^2} \ . \tag{10.69}$$

For now let us write informally the prior probability $\mathcal{P}[f]$ as

$$\mathcal{P}[f] \propto e^{-\|f\|_k^2} \ . \tag{10.70}$$

Following the Bayes rule (10.68) the *a posteriori* probability of f is written as

$$\mathcal{P}[f|X \times Y] \propto e^{-[\frac{1}{2\sigma^2}\sum_{i=1}(y_i-f(\mathbf{x}_i))^2+\|f\|_k^2]} \ . \tag{10.71}$$

One of the several possible estimates [Marroquin et al., 1987] of the function f from the probability distribution (10.71) is the so called MAP (*Maximum A Posteriori*) estimate, that considers the function that maximizes the *a posteriori* probability $\mathcal{P}[f|X \times Y]$, and therefore minimizes the exponent in equation (10.71). The MAP estimate of f is therefore the minimizer of the functional:

$$\frac{1}{m}\sum_{i=1}(y_i - f(\mathbf{x}_i))^2 + \frac{1}{m}\Lambda\|f\|_k^2 \tag{10.72}$$

where Λ is the *a priori* defined constant $2\sigma^2$, that is

$$\frac{1}{m}\sum_{i=1}(y_i - f(\mathbf{x}_i))^2 + \tilde{\lambda}\|f\|_k^2 \ . \tag{10.73}$$

where $\tilde{\lambda} = \frac{\Lambda}{m}$. This functional is the same as that of equation (10.67), but here it is important to notice that $\lambda(m) = \frac{\Lambda}{m}$. As noticed by Girosi et al. [1995], functionals of the type (10.70) are common in statistical physics [Parisi, 1988], where the stabilizer (here $\|f\|_k^2$) plays the role of an energy functional. As we will see later, the RKHS setting we use in this chapter makes clear that the correlation function of the physical system described by $\|f\|_k^2$ is the kernel $k(\mathbf{x}, \mathbf{y})$.[16]

Thus in the standard MAP interpretation of RN the data term is a model of the noise and the stabilizer is a prior on the regression function f. The informal argument outlined above can be made formally precise in the setting of this chapter in which the stabilizer is a norm in a RKHS (see also [Wahba, 1990]). To see the argument in more detail, let us write the prior (10.70) as:

$$P[f] \propto e^{-\|f\|_k^2} = e^{-\sum_{n=1}\frac{a_n^2}{\lambda_n}} \tag{10.74}$$

16. As observed by Girosi et al. [1995], (see also [Poggio and Girosi, 1989]) prior probabilities can also be seen as a measure of complexity, assigning high complexity to the functions with small probability. It has been proposed by Rissanen [1978] to measure the complexity of a hypothesis in terms of the bit length needed to encode it. It turns out that the MAP estimate mentioned above is closely related to the Minimum Description Length Principle: the hypothesis f which for given $X \times Y$ can be described in the most compact way is chosen as the "best" hypothesis. Similar ideas have been explored by others (see [Vapnik, 1995, 1998] for a summary).

where M is the dimensionality of the RKHS, with possibly $M = \infty$. Of course functions f can be represented as vectors $\mathbf{a}$ in the reference system of the eigenfunctions ψ_n of the kernel k since

$$f(\mathbf{x}) = \sum_{n=1} a_n \psi_n(\mathbf{x}) \ . \tag{10.75}$$

The stabilizer

$$\|f\|_k^2 = \sum_{n=1} \frac{a_n^2}{\lambda_n} = \mathbf{a}^T \lambda^{-1} \mathbf{a} \tag{10.76}$$

can of course be also expressed in any other reference system ($\psi' = A\psi$) as

$$\|f\|_k^2 = \mathbf{b}^T K^{-1} \mathbf{b} \tag{10.77}$$

which suggests that K can be interpreted as the covariance matrix in the reference system of the ψ'. It is clear in this setting that the stabilizer can be regarded as the Malahanobis distance of f from the mean of the functions. $P[f]$ is therefore a multivariate Gaussian with zero mean in the Hilbert space of functions defined by k and spanned by the ψ_n:

$$P[f] \propto e^{-\|f\|_k^2} = e^{-(\mathbf{b}^T K^{-1} \mathbf{b})} \ . \tag{10.78}$$

Thus the stabilizer can be related to a Gaussian prior on the function space.

The interpretation is attractive since it seems to capture the idea that the stabilizer effectively constrains the desired function to be in the RKHS defined by the kernel k. It also seems to apply not only to standard regularization but to any functional of the form

$$R_{\text{reg}}[f] = \frac{1}{m} \sum_{i=1} V(y_i - f(\mathbf{x}_i)) + \lambda \|f\|_k^2 \tag{10.79}$$

where $V(\cdot)$ is any monotonically increasing loss function (see [Girosi et al., 1991]). In particular it can be applied to the SVM (regression) case in which the relevant functional is

$$\frac{1}{m} \sum_{i=1} |y_i - f(\mathbf{x}_i)|_\epsilon + \lambda \|f\|_k^2 \ . \tag{10.80}$$

In both cases, one can write $P[X \times Y|f]$ and $P[f]$ for which the MAP estimate of

$$P[f|X \times Y] \propto P[X \times Y|f] P[f] \tag{10.81}$$

gives either equation (10.79) or equation (10.80). Of course, the MAP estimate is only one of several possible. In many cases, the average of $f = \int f dP[f|X \times Y]$ may make more sense[17] (see [Marroquin et al., 1987]). This argument provides a formal proof of the well-known equivalence between Gaussian processes defined by

17. In the Gaussian case - Regularization Networks - MAP and average estimates coincide.

the previous equation with $P[f|X \times Y]$ Gaussian and the RN defined by (10.67).[18] In the following we comment separately on the stabilizer – common to RN and SVM – and on the data term – which is different in the two cases.

10.6.2 Bayesian Interpretation of the Stabilizer in the RN and SVM Functionals

Assume that the problem is to estimate f from sparse data y_i at location $\mathbf{x}_i$. From the previous description it is clear that choosing a kernel k is equivalent to assuming a Gaussian prior on f with covariance equal to k. Thus choosing a prior through k is equivalent a) to assume a Gaussian prior and b) to assume a correlation function associated with the family of functions f. The relation between positive definite kernels and correlation functions k of Gaussian random processes is characterized in details in [Wahba, 1990, Theorem 5.2]. In applications it is natural to use an empirical estimate of the correlation function, whenever available. Notice that in the MAP interpretation a Gaussian prior is *assumed* in RN as well as in SVM. For both RN and SVM when empirical data are available on the statistics of the family of functions of the form (10.75) one should check that $P[f]$ is Gaussian and make it zero-mean. Then an empirical estimate of the correlation function $E[f(\mathbf{x})f(\mathbf{y})]$ (with the expectation relative to the distribution $P[f]$) can be used as the kernel. Notice also that the basis functions ψ_n associated with the positive definite function $k(\mathbf{x}, \mathbf{y})$ correspond to the Principal Components associated with k.

10.6.3 Bayesian Interpretation of the Data Term in the Regularization and SVM Functional

As already observed the model of the noise that has to be associated with the data term of the SVM functional is *not* Gaussian additive as in RN. The same is true for the specific form of Basis Pursuit Denoising considered in Section 10.7, given the equivalence with SVM. Data terms of the type $V(y_i - f(\mathbf{x}_i))$ can be interpreted [Girosi et al., 1991] in probabilistic terms as non-Gaussian noise models. Recently, Pontil et al. [1998a] derived the noise model corresponding to Vapnik's ϵ-insensitive loss function. It turns out that the underlying noise model consists of the superposition of Gaussian processes with different variances and means, that is[19]:

$$\exp\left(-|x|_\epsilon\right) = \int_{-\infty}^{+\infty} dt \int_0^\infty d\beta \lambda(t) \mu(\beta) \sqrt{\beta} \exp\left(-\beta(x-t)^2\right), \tag{10.82}$$

18. Ironically, it is only recently that the neural network community seems to have realized the equivalence of many so-called neural networks and Gaussian processes and the fact that they work quite well (see [MacKay, 1997, Williams, 1998] and references therein).
19. In the following we introduce the variable $\beta = (2\sigma^2)^{-1}$.

with:

$$\lambda_\epsilon(t) = \frac{1}{2(\epsilon+1)}\left(\chi_{[-\epsilon,\epsilon]}(t) + \delta(t-\epsilon) + \delta(t+\epsilon)\right), \tag{10.83}$$

$$\mu(\beta) \propto \beta^2 \exp\left(-\frac{1}{4\beta}\right). \tag{10.84}$$

where $\chi_{[-\epsilon,\epsilon]}(t)$ is 1 for $t \in [-\epsilon,\epsilon]$, 0 otherwise. For the derivation see [Pontil et al., 1998a]. Notice that the variance has a unimodal distribution that does not depend on ϵ, and the mean has a distribution which is uniform in the interval $[-\epsilon,\epsilon]$, (except for two delta functions at $\pm\epsilon$, which ensures that the mean has not zero probability to be equal to $\pm\epsilon$). The distribution of the mean is consistent with the current understanding of Vapnik's ILF: errors smaller than ϵ do not count because they may be due entirely to the bias of the Gaussian noise.

10.6.4 Why a MAP Interpretation may be Misleading

We have just seen that minimization of both the RN and the SVMR functionals can be interpreted as corresponding to the MAP estimate of the posterior probability of f given the data, for certain models of the noise and for a specific Gaussian prior on the space of functions f. However, a Bayesian interpretation of this type may in general be *inconsistent* with Structural Risk Minimization and more generally with Vapnik's analysis of the learning problem. The following argument due to Vapnik shows the general point.

Consider functionals (10.35) and (10.53). From a Bayesian point of view instead of the parameter λ – which in RN and SVM is a function of the data (through the SRM principle) – we have $\tilde{\lambda}$ which depends on the data as $\frac{\Lambda}{m}$: the constant Λ has to be independent of the training data (i.e., their size m). On the other hand, as we discussed in Section 10.2, SRM dictates a choice of λ depending on the training set. It seems unlikely that λ could simply depend on $\frac{\Lambda}{m}$ as the MAP interpretation requires for consistency.

Fundamentally, the core of Vapnik's analysis is that the key to learning from finite training sets is *capacity control*, that is the control of the complexity of the hypothesis space as a function of the training set. From this point of view the ability to choose λ as a function of the training data is essential to our interpretation of Regularization and SVM in terms of the VC theory (compare the procedure described in our SRM section 10.2). Full capacity control and appropriate dependency of λ on the training set, which we expect in the general case not to be simply of the form $\frac{\Lambda}{m}$, is lost in the direct MAP interpretation that we described in this chapter. Of course, an empirical Bayesian interpretation relying on hyper-parameters in the prior is possible and often useful but it amounts to little more than a parametric form for the posterior distribution, usually used in conjunction with maximum likelihood estimation of the parameters from the data.

10.7 Connections Between SVMs and Sparse Approximation Techniques

In recent years there has been a growing interest in approximating functions and representing signals using linear superposition of a small number of basis functions selected from a large, redundant set of basis functions, called a *dictionary*. These techniques go under the name of Sparse Approximations (SAs) [Chen, 1995, Chen et al., 1999, Olshausen and Field, 1996, Harpur and Prager, 1996, Daubechies, 1992, Mallat and Zhang, 1993, Coifman and Wickerhauser, 1992, DeVore, 1998]. We will start with a short overview of SAs. Then we will discuss a result due to Girosi [1998] that shows an equivalence between SVMs and a particular SA technique.

10.7.1 The Problem of Sparsity

Given a dictionary of basis functions (for example a frame, or just a redundant set of basis functions) $\{\psi_1(\mathbf{x}), \ldots, \psi_n(\mathbf{x})\}$ with n very large (possibly infinite), SA techniques seek an approximation of a function $f(\mathbf{x})$ as a linear combination of the smallest number of elements of the dictionary, that is, an approximation of the form:

$$f_\alpha(\mathbf{x}) = \sum_{j=1}^{j} \alpha_j \psi_j(\mathbf{x}), \tag{10.85}$$

with the smallest number of non-zero coefficients α_i. Formally, the problem is formulated as minimizing the following cost function:

$$\mathcal{R}[\boldsymbol{\alpha}] = D\left(f(\mathbf{x}), \sum_{j=1}^{n} \alpha_j \psi_j(\mathbf{x})\right) + \epsilon \|\boldsymbol{\alpha}\|_{L_0}, \tag{10.86}$$

where D is a cost measuring the distance (in some predefined norm) between the true function $f(\mathbf{x})$ and our approximation, the L_0 norm of a vector counts the number of elements of that vector which are different from zero, and ϵ is a parameter that controls the trade off between sparsity and approximation. Observe that the larger ϵ is in (10.86), the more sparse the solution will be.

In the more general case of learning function f is not given, and instead we have a data set $X \times Y = \{(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_m, y_m)\}$ of the values y_i of f at locations $\mathbf{x}_i$.[20] Note that in order to minimize $\mathcal{R}[\boldsymbol{\alpha}]$ we need to know f at all points $\mathbf{x}$. In the learning paradigm, in the particular case that $D(f(\mathbf{x}), \sum_{j=1}^{n} \alpha_j \psi_j(\mathbf{x})) = \|f(\mathbf{x}) - \sum_{j=1}^{n} \alpha_j \psi_j(\mathbf{x})\|_{L_2}^2$, the first term in equation (10.86) is replaced by an empirical one, and (10.86) becomes:

$$\frac{1}{m} \sum_{i=1} \left(y_i - \sum_{j=1}^{n} \alpha_j \psi_j(\mathbf{x_i})\right)^2 + \epsilon \|\boldsymbol{\alpha}\|_{L_0} \tag{10.87}$$

20. For simplicity we consider the case where the input distribution $P(\mathbf{x})$ is distribution.

Minimizing (10.86) can be used as well to find sparse approximations in the case that the function f is generated by a function f_0 corrupted by additive noise. In this case the problem can be formulated as finding a solution $\boldsymbol{\alpha}$ to:

$$f = \Psi\boldsymbol{\alpha} + \eta \tag{10.88}$$

with the smallest number of non-zero elements, where Ψ is the matrix with columns the elements of the dictionary, and η is the noise. If we take a probabilistic approach and the noise is Gaussian, the problem can again be formulated as minimizing:

$$\mathcal{R}[\boldsymbol{\alpha}] = \left\| f(\mathbf{x}) - \sum_{j=1}^{n} \alpha_j \psi_j(\mathbf{x}) \right\|_{L_2}^2 + \epsilon \|\boldsymbol{\alpha}\|_{L_0}, \tag{10.89}$$

Unfortunately it can be shown that minimizing (10.86) is NP-hard because of the L_0 norm. In order to circumvent this shortcoming, approximated versions of the cost function above have been proposed. For example, Chen [1995], Chen et al. [1999] use the L_1 norm as an approximation of the L_0 norm, obtaining an approximation scheme that they call *Basis Pursuit De-Noising* (BPDN) which consists of minimizing:

$$\mathcal{R}[\boldsymbol{\alpha}] = \left\| f(\mathbf{x}) - \sum_{j=1}^{n} \alpha_j \psi_j(\mathbf{x}) \right\|_{L_2}^2 + \epsilon \sum_{j=1}^{n} |\alpha_j|, \tag{10.90}$$

10.7.2 Equivalence between BPDN and SVMs

In this section we consider the particular case in which we are given a data set $X \times Y = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, and the dictionary consists of basis functions of the form:

$$\psi_j(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}_i) \quad \forall i = 1, \ldots, m \tag{10.91}$$

where k is the reproducing kernel of a RKHS $\mathcal{H}_k$, and the size m of $X \times Y$ is equal to the size n of the dictionary. Moreover, following Girosi [1998], we assume that $f(\mathbf{x})$ in eq. (10.86) is in the RKHS, and we use as the cost D in (10.86) the norm in the RKHS $\mathcal{H}_k$ induced by the kernel k, and approximate the L_0 norm with L_1. Under these assumptions, we get the SA technique that minimizes:

$$\mathcal{R}[\boldsymbol{\alpha}] = \left\| f(\mathbf{x}) - \sum_{j=1} \alpha_j \psi_j(\mathbf{x}) \right\|_{k}^2 + \epsilon \|\boldsymbol{\alpha}\|_{L_1}. \tag{10.92}$$

subject to $f(\mathbf{x}_i) = y_i$.

It can be shown [Girosi, 1998] that this technique is equivalent to SVMR in the following sense: the two techniques give the same solution, which is obtained by solving the same quadratic programming problem. Girosi [1998] proves the equivalence between SVMR and BPDN under the assumption that the data set $\{(\mathbf{x}_i, y_i)\}_{i=1}$ has been obtained by sampling, *in absence of noise*, the target function

f. Functional (10.92) differs from (10.90) only in the cost D. While Chen et al. [1999], in their BPDN method, measure the reconstruction error with an L_2 criterion, Girosi measures it by the true distance, in the $\mathcal{H}_k$ norm, between the target function f and the approximating function f^*. This measure of distance, which is common in approximation theory, is better motivated than the L_2 norm because it not only enforces closeness between the target and the model, but also between their derivatives, since $\|\cdot\|_k$ is a measure of smoothness.

Notice that from eq. (10.92) the cost function E cannot be computed because it requires the knowledge of f (in the first term). If we had $\|\cdot\|_{L_2}$ instead of $\|\cdot\|_k$ in eq. (10.92),this would force us to consider the approximation:

$$\|f(\mathbf{x}) - f^*(\mathbf{x})\|_{L_2}^2 \approx \frac{1}{m}\sum_{i=1}(y_i - f^*(\mathbf{x}_i))^2 \tag{10.93}$$

However if we used the norm $\|\cdot\|_k$ we can use the reproducing property obtaining (see [Girosi, 1998]):

$$\mathcal{R}[\boldsymbol{\alpha}] = \frac{1}{2}\left(\|f\|_k^2 + \sum_{i,j=1}\alpha_i\alpha_j k(\mathbf{x}_i,\mathbf{x}_j) - 2\sum_{i=1}\alpha_i y_i\right) + \epsilon\|\boldsymbol{\alpha}\|_{L_1} \tag{10.94}$$

Observe that functional (10.94) is the same as the objective function of SVM of problem 10.8 up to the constant $\frac{1}{2}\|f\|_k^2$. However, in the SVM formulation the coefficients α_i satisfy two constraints, which in the case of sparsity are trivially satisfied under further assumptions. For details see [Girosi, 1998]. It also follows from eq. (10.85) and (10.91) that the approximating function is of the form:

$$f^*(\mathbf{x}) \equiv f_{\boldsymbol{\alpha}}(\mathbf{x}) = \sum_{i=1}\alpha_i k(\mathbf{x};\mathbf{x}_i). \tag{10.95}$$

This model is similar to the one of SVM (eq. (10.55)), except for the constant b. This relation between SVMR and SA suggests directly that SVM yield a sparse representation.

10.8 Remarks

10.8.1 Regularization Networks can implement SRM

One of the main focuses of this review is to describe and motivate the classical technique of regularization – minimization of functionals such as in equation (10.1) – within the framework of VC theory. In particular we have shown that classical regularization functionals can be motivated within the statistical framework of capacity control.

Standard Regularization	$R_{\rm reg}[f] = \frac{1}{m}\sum_{i=1}(y_i - f(\mathbf{x}_i))^2 + \lambda\|f\|_k^2$
SVM Regression (SVMR)	$R_{\rm reg}[f] = \frac{1}{m}\sum_{i=1}\|y_i - f(\mathbf{x}_i)\|_\epsilon + \lambda\|f\|_k^2$
SVM Classification (SVMC)	$R_{\rm reg}[f] = \frac{1}{m}\sum_{i=1}\theta(1 - y_i f(\mathbf{x}_i))(1 - y_i f(\mathbf{x}_i)) + \lambda\|f\|_k^2$

Table 10.2 A unified framework: the minimizer of each of these three functionals has always the same form: $f(\mathbf{x}) = \sum_{i=1}\alpha_i k(\mathbf{x}, \mathbf{x}_i)$ or $f(\mathbf{x}) = \sum_{i=1}\alpha_i k(\mathbf{x}, \mathbf{x}_i) + b$. Of course in classification the decision function is $\mathrm{sgn}\,(f(\mathbf{x}))$.

10.8.2 The SVM Functional is a Special Formulation of Regularization

Throughout our review it is clear that standard Regularization Networks as well as Support Vector Machines for regression and Support Vector Machines for classification (see Table 10.2) can be justified within the same framework, based on Vapnik's SRM principle and the notion of V_γ dimension. The three functionals of the table have different loss functions $c(\cdot,\cdot,\cdot)$ but the same stabilizer. Thus the minimizer has the same general form and, as a consequence, the associated network has the same architecture. In particular, RKHS, associated kernels, and the mapping they induce from the input space into a higher dimensional space of features ψ_n, are exactly the same in SVM as in RN. The different loss functions of SVM determine however quite different properties of the solution (see Table (10.2)) which is, unlike regularization, sparse in the α_n. Notice that loss functions different from quadratic loss have been used before in the context of regularization. In particular, the physical analogy of representing the data term using nonlinear spring (standard L_2 regularization corresponds to linear springs) was used and studied before (for instance see [Girosi et al., 1991]). It is, however, the specific choice of the loss functions in SVMC and SVMR that provides some of their characteristic features, such as sparsity of the solution. Notice also that the geometric interpretation of $\|f\|_k^2$ in terms of the *margin* [Vapnik, 1998] is true only for the classification case and depends on the specific loss function $c(\cdot,\cdot,\cdot)$ used in SVMC.

10.8.3 Capacity Control and the Physical World

An interesting question outside the realm of mathematics which has been asked recently is why large margin classifiers seem to work well enough in the physical world. As we saw throughout this review, the question is really the same as the question of why to assume smoothness in regression, that is why to use stabilizers such as $\|f\|_k^2$, which are usually smoothness functionals. Smoothness can be justified by observing that in many cases smoothness of input-output relations are implied directly by the existence of physical laws with continuity and differentiability properties. In classification minimization of $\|f\|_k^2$ corresponds to maximization of the margin in the space of the ψ_n; it is also equivalent to choosing the decision boundary resulting from thresholding the smoothest f in the

original space, according to the smoothness criterion induced by k (notice that the decision boundary is the level crossing of f and not necessarily smooth everywhere). Conversely, we would not be able to generalize for input-output relations that are not smooth, that is for which "similar" inputs do not correspond to "similar" outputs (in an appropriate metric!). Such cases exist: for instance the mapping provided by a telephone directory between names and telephone numbers is usually not "smooth" and it is a safe bet that it would be difficult to learn it from examples. In cases in which physical systems are involved, however, input-output relations have some degree of smoothness and can be learned. From this point of view large margin (in feature space) and smoothness are properties of the physical world that are key to allow generalization, learning and the development of theories and models.

Acknowledgments

We would like to thank for suggestions of Chris Burges, Peter Bartlett, Nello Cristianini, Grace Wahba and Bernhard Schölkopf. We are grateful for many discussions with Alessandro Verri, Sayan Mukherjee and Rif Rifkin. Very special thanks go to Federico Girosi and Vladimir Vapnik.

III Boosting

11 Robust Ensemble Learning

Gunnar Rätsch
GMD FIRST
Kekuléstr. 7
12489 Berlin, Germany
raetsch@first.gmd.de

Bernhard Schölkopf
Microsoft Research Limited
St. George House, 1 Guildhall Street
Cambridge CB2 3NH, UK
bsc@microsoft.com

Alexander J. Smola
Department of Engineering
Australian National University
Canberra ACT 0200, Australia
Alex.Smola@anu.edu.au

Sebastian Mika
GMD FIRST
Kekuléstr. 7
12489 Berlin, Germany
mika@first.gmd.de

Takashi Onoda
CRIEPI
2-11-1, Iwado Kita, Komae-shi
Tokyo, Japan
onoda@criepi.denken.or.jp

Klaus–Robert Müller
GMD FIRST
Kekuléstr. 7
12489 Berlin, Germany
klaus@first.gmd.de

AdaBoost and other ensemble methods have successfully been applied to a number of classification tasks, seemingly defying problems of overfitting. AdaBoost performs gradient descent in an error function with respect to the margin, asymptotically concentrating on the patterns which are hardest to learn. For noisy problems, however, this can be disadvantageous. Indeed, theoretical analysis has shown that the margin *distribution*, as opposed to just the minimal margin, plays a crucial role in understanding this phenomenon. Loosely speaking, some outliers should be tolerated if this has the benefit of substantially increasing the margin on the remaining points.

We propose new boosting algorithms which, similar to ν-Support-Vector Classification, allows for the possibility of a pre-specified fraction ν of points to lie in the margin area or even on the wrong side of the decision boundary.

Unlike other regularized boosting algorithms [Mason et al., 1999, Rätsch et al., 1998], this gives a nicely interpretable way of controlling the trade-off between minimizing the training error and capacity.

11.1 Introduction

Boosting and related Ensemble learning methods have been recently used with great success in applications such as Optical Character Recognition [Drucker et al., 1993, LeCun et al., 1995, Schwenk and Bengio, 1998] (see also Section 1.5).

The idea of a large (minimum) margin explains the good generalization performance of AdaBoost in the low noise regime. However, AdaBoost performs worse on noisy tasks [Quinlan, 1996b, Rätsch et al., 1998], such as the *iris* and the *breast cancer* benchmark data sets [Blake et al., 1998]. On the latter tasks, a large margin on *all* training points cannot be achieved without adverse effects on the generalization error. This experimental observation was supported by the study of Schapire et al. [1998] (see also Theorem 1.15), where the generalization error of ensemble methods was bounded by the sum of the fraction of training points which have a margin smaller than some value ρ, say, plus a complexity term depending on the base hypotheses and ρ. While this worst-case bound[1] can only capture part of what is going on in practice, it nevertheless already conveys the message that in some cases it pays to allow for some points which have a small margin, or are misclassified, if this leads to a larger overall margin on the remaining points.

To cope with this problem, it was mandatory to construct *regularized* variants of AdaBoost, which traded off the number of margin errors and the size of the margin [Mason et al., 1999, Rätsch et al., 1998]. This goal, however, was so far achieved in a heuristic way by introducing regularization parameters which have no immediate interpretation and which cannot be adjusted easily.

1. Note that the complexity term depends on the VC dimension of the base hypothesis, which is a worst case capacity measure.

The present chapter addresses this problem in two ways. Primarily, it makes an *algorithmic* contribution (including pseudocode) to the problem of constructing regularized boosting algorithms. However, compared to the previous efforts, it parameterizes the trade-off in a much more intuitive way: its only free parameter directly determines the fraction of margin errors.

This, in turn, is also appealing from a *theoretical* point of view, since it involves a parameter which controls a quantity that plays a crucial role in the generalization error bounds (cf. also Mason et al. [1999], Schapire et al. [1998]). Furthermore, it allows the user to roughly specify this parameter once a reasonable estimate of the expected error (possibly from other studies) can be obtained, thus reducing the training time.

11.2 Boosting and the Linear Programming Solution

Before deriving new algorithms, we briefly discuss the properties of the solution generated by standard AdaBoost and, closely related, Arc-GV [Breiman, 1999], and discuss the relation to a linear programming (LP) solution over the class of base hypotheses G. Let us recall the definitions from Section 1.4: Let $\{g_t(\mathbf{x}) : t = 1, \ldots, T\}$ be a sequence of hypotheses and $\boldsymbol{\alpha} = [\alpha_1 \ldots \alpha_T]$ their weights satisfying $\alpha_t \geq 0$. The hypotheses g_t are elements of a hypotheses class[2] $G = \{g : \mathbf{x} \mapsto \{\pm 1\}\}$, which is defined by a base learning algorithm L.

The ensemble generates the label which is the weighted majority of the votes by $\operatorname{sgn}(f(\mathbf{x}))$ where

$$f(\mathbf{x}) = \sum_t \frac{\alpha_t}{\|\boldsymbol{\alpha}\|_1} g_t(\mathbf{x}). \tag{11.1}$$

margins

In order to express that f and therefore also the margin ρ depend on $\boldsymbol{\alpha}$ and for the ease of notation we define (cf. also Definition 1.2)

$$\rho(\mathbf{z}, \boldsymbol{\alpha}) := y f(\mathbf{x}) \text{ where } \mathbf{z} := (\mathbf{x}, y) \text{ and } f \text{ is defined as in (11.1).} \tag{11.2}$$

Likewise we set

$$\rho(\boldsymbol{\alpha}) := \min_{1 \leq i \leq m} \rho(\mathbf{z}_i, \boldsymbol{\alpha}) \ , \tag{11.3}$$

i.e., we will use the *normalized* margin.

minimization objective

The minimization objective of AdaBoost can be expressed in terms of the margins

$$\mathcal{G}(\boldsymbol{\alpha}) := \sum_{i=1}^{m} \exp(-\|\boldsymbol{\alpha}\|_1 \rho(\mathbf{z}_i, \boldsymbol{\alpha})) \ . \tag{11.4}$$

2. Most of the work is also applicable to a hypothesis class of the type $G = \{g : \mathbf{x} \mapsto [-1, 1]\}$, except for all things which depend on the definition of ϵ_t.

In every iteration AdaBoost tries to minimize this error by a stepwise maximization of the margin. It is widely believed that AdaBoost tries to maximize the *smallest margin* on the training set [Breiman, 1999, Freund and Schapire, 1997, Frean and Downs, 1998, Schapire et al., 1998, Rätsch et al., 1998]. Strictly speaking, however, a general proof is missing. It would imply that AdaBoost asymptotically approximates (up to scaling) the solution of the following linear programming problem over the complete hypothesis set G (cf. Grove and Schuurmans [1998], assuming a *finite* number of basis hypotheses):

linear programming

$$\begin{array}{lrll} \text{maximize} & & \rho & \\ \text{subject to} & \rho(\mathbf{z}_i, \boldsymbol{\alpha}) & \geq \rho & \text{for all } 1 \leq i \leq m \\ & \alpha_t, \rho & \geq 0 & \text{for all } 1 \leq t \leq |G| \\ & \|\boldsymbol{\alpha}\|_1 & = 1 & \end{array} \tag{11.5}$$

Since such a linear program cannot be solved exactly for a infinite hypothesis set in general, it is interesting to analyze approximation algorithms for this kind of problems. First we have a look at the asymptotic properties of two Boosting algorithms (AdaBoost and Arc-GV).

Considering the optimization strategy of AdaBoost, it essentially consists of two parts: (i) the selection of a hypothesis and then (ii) the weighting of this hypothesis. In turn, we address their benefits (or shortcomings, respectively) for achieving convergence to the LP solution.

choosing hypotheses

The first part is done by adaptively re-weighting the patterns in each iteration t

$$w_{t+1}(\mathbf{z}_i) = \frac{\partial \mathcal{G}(\boldsymbol{\alpha}^t)/\partial \rho(\mathbf{z}_i, \boldsymbol{\alpha}^t)}{\sum_{j=1}^m \partial \mathcal{G}(\boldsymbol{\alpha}^t)/\partial \rho(\mathbf{z}_j, \boldsymbol{\alpha}^t)} = \frac{\exp(-\|\boldsymbol{\alpha}^t\|_1 \rho(\mathbf{z}_i, \boldsymbol{\alpha}^t))}{\sum_{j=1}^m \exp(-\|\boldsymbol{\alpha}^t\|_1 \rho(\mathbf{z}_j, \boldsymbol{\alpha}^t))}, \tag{11.6}$$

where $\boldsymbol{\alpha}^t = [\alpha_1, \ldots, \alpha_t, 0, 0, \ldots]$ is the weight vector after the t-th iteration. AdaBoost assigns the highest weights to the patterns with the smallest margin, i.e., to those patterns for which the inequality on the margin in (11.5) is (almost) an equality. In the case of an equality for a pattern in (11.5) the Lagrange multiplier would be non-zero (if $\boldsymbol{\alpha}$ is optimal). Increasing the weight will eventually lead to a larger margin of the corresponding pattern and, hence, to an increase of ρ. This way of finding a new hypothesis seems to be appropriate. After all, Arc-GV, which has been proven to converge [Breiman, 1999], is identical to AdaBoost in the first part (cf. page 212), i.e., in choosing the hypotheses.

choosing coefficients

The second part is computing the weight of the selected hypothesis α_t. We first consider AdaBoost. Then we show what has to be changed to get Arc-GV and the desired convergence property.

AdaBoost minimizes the function $\mathcal{G}$ with respect to the weight α_t for the new hypothesis. However, minimizing $\mathcal{G}$ does not guarantee maximization of $\rho(\boldsymbol{\alpha})$ (cf. Figure 11.1). In particular, in the t-th iteration, the minimization of $\mathcal{G}$ wrt. α_t amounts to:

$$\alpha_t := \operatorname*{argmin}_{\alpha_t \geq 0} \sum_{i=1}^m \exp\left[-(\|\boldsymbol{\alpha}^{t-1}\|_1 + \alpha_t)\rho\left(\mathbf{z}_i, \boldsymbol{\alpha}^t\right)\right] \tag{11.7}$$

$$= \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right), \tag{11.8}$$

where ϵ_t is defined as in Algorithm 1.2 (cf. footnote 2). Now suppose that $\|\boldsymbol{\alpha}^{t-1}\|_1$ is large enough and $\rho(\boldsymbol{\alpha}^t) > 0$. Then $\mathcal{G}$ may be minimized either by maximizing the margin (which brings us closer to the LP solution) or by increasing $\|\boldsymbol{\alpha}^t\|_1 = \|\boldsymbol{\alpha}^{t-1}\|_1 + \alpha_t$ (which increases the slope of the cost function). Therefore, in this case, one will obtain values of α_t somewhat larger than the value maximizing ρ (see Figure 11.1 for an example). In other cases it may occur that α_t is chosen too small.

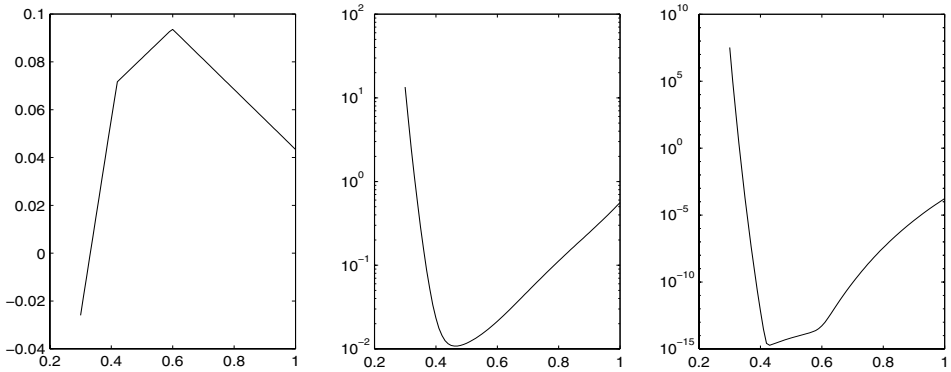


Figure 11.1 Illustration of the non-optimal selection of α_t in (11.7) (on a toy example): Left: $\rho(\boldsymbol{\alpha})$; middle and right: $\mathcal{G}(\boldsymbol{\alpha})$ for two different stages in the boosting process ($\|\boldsymbol{\alpha}\|_1 = 30$ and 300). We set $\boldsymbol{\alpha} = \boldsymbol{\alpha} + (1-\lambda)\lambda^{-1}\|\boldsymbol{\alpha}\|_1\mathbf{e}_t$, where $\mathbf{e}_t$ being the t-th unit vector. In all three cases, λ is shown on the abscissae. Clearly, the maximum of ρ and the minimum of G do not match.

A different, particularly simple strategy would be to choose α_t such that the margin is maximal, i.e.,

$$\alpha_t := \operatorname*{argmax}_{\alpha_t>0} \min_{1\le i\le m} \rho(\mathbf{z}_i, \boldsymbol{\alpha}^t).$$

Unfortunately, there exist examples where this strategy leads to non-optimal margins: Assume we have $n > 2$ hypotheses and $m > 2$ patterns. Now we run the algorithm and get at some iteration the margins $[\rho(\mathbf{z}_1), \rho(\mathbf{z}_2), \ldots, \rho(\mathbf{z}_m)] := [y_1 f(\mathbf{x}_1), y_2 f(\mathbf{x}_2), \ldots, y_m f(\mathbf{x}_m)]$ with $\rho(\mathbf{z}_1) = \rho(\mathbf{z}_2) < \min(\rho^{\mathrm{lp}}, \rho(\mathbf{z}_i))$, $(i = 3, \ldots, m)$, on the training set. Furthermore, suppose two hypotheses have the margins $[-1, 1, \ldots]$ and $[1, -1, \ldots]$. If one of these is chosen, then α_t will be 0, but there exists a solution such that $\rho(\boldsymbol{\alpha}) > \rho(\mathbf{z}_1)$ using all n hypotheses. (The rest of the hypotheses can be constructed in such a way that there exists no hypothesis which would increase the margin.) Interestingly, AdaBoost would choose $\alpha_t > 0$ and would eventually be able to solve this dilemma (for $\rho(\mathbf{z}_1) > 0$).

Breiman [1999] proposed a modification of AdaBoost – Arc-GV – making it possible to show the asymptotic convergence of $\rho(\boldsymbol{\alpha}^t)$ to the global solution ρ^{lp}:

Breiman's convergence results

Theorem 11.1 [Breiman, 1999]
Choose α_t in each iteration as

$$\alpha_t := \underset{\alpha\in[0,1]}{\operatorname{argmin}} \sum_{i=1}^{m} \exp\left[-\|\boldsymbol{\alpha}^t\|_1 \left(\rho(\mathbf{z}_i, \boldsymbol{\alpha}^t) - \rho(\boldsymbol{\alpha}^{t-1})\right)\right], \tag{11.9}$$

and assume that the base learner always finds the hypothesis $g \in G$, which minimizes the weighted training error with respect to the weights given in (11.6). Then

$$\lim_{t\to\infty} \rho(\boldsymbol{\alpha}^t) = \rho^{\text{lp}} \ ,$$

where ρ^{lp} is the maximum possible margin for a combined classifier from G.

Arc-GV's objective

Note that the algorithm above can be derived from a slightly modified error function:

$$\mathcal{G}_{\text{gv}}(\boldsymbol{\alpha}^t) := \sum_{i=1}^{m} \exp\left[-\|\boldsymbol{\alpha}^t\|_1 \left(\rho(\mathbf{z}_i, \boldsymbol{\alpha}^t) - \rho(\boldsymbol{\alpha}^{t-1})\right)\right]. \tag{11.10}$$

The optimization problem in (11.9) can be solved analytically and we get [Breiman, 1999]

$$\alpha_t = \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right) + \log\left(\frac{1-\rho(\boldsymbol{\alpha}^{t-1})}{1+\rho(\boldsymbol{\alpha}^{t-1})}\right) \ . \tag{11.11}$$

Thus, we just have an additional term which only depends on the margin in the last iteration. It always has the opposite sign as $\rho(\boldsymbol{\alpha}^{t-1})$.

The question one might ask now is, whether to use AdaBoost or Arc-GV in practice, or more specifically, whether Arc-GV converges fast enough to profit from its asymptotic properties? To this end, we conduct an experiment with about 25 (appropriate) hypotheses on a data set with 50 patterns.[3] In this finite hypothesis class setting we can easily solve the linear program maximizing the margin and can expect that AdaBoost and also Arc-GV reach the asymptotic regime after a reasonable number of iterations. First, we generate appropriate hypotheses and data sets, i.e., learning problems, such that we get a selection of ρ^{lp} (some negative and some positive). Then we run AdaBoost and Arc-GV on these sets and record $\rho(\boldsymbol{\alpha}^t)$ after 100 and 1000 iterations. Figure 11.2 shows the results of this experiment and we can observe that (a) AdaBoost has problems finding the optimal combination if $\rho^{\text{lp}} < 0$, (b) Arc-GV's convergence does not depend on ρ^{lp}, and (c) for $\rho^{\text{lp}} > 0$ AdaBoost usually converges to the maximum margin solution faster than Arc-GV.

toy experiment

observations

Observation (a) becomes clear from (11.4): $\mathcal{G}(\boldsymbol{\alpha})$ will not converge to 0 and $\|\boldsymbol{\alpha}\|_1$ can be bounded by some value. Thus the asymptotic case cannot be reached,

3. With the `banana shape` dataset from http://www.first.gmd.de/~raetsch/data/banana.html.

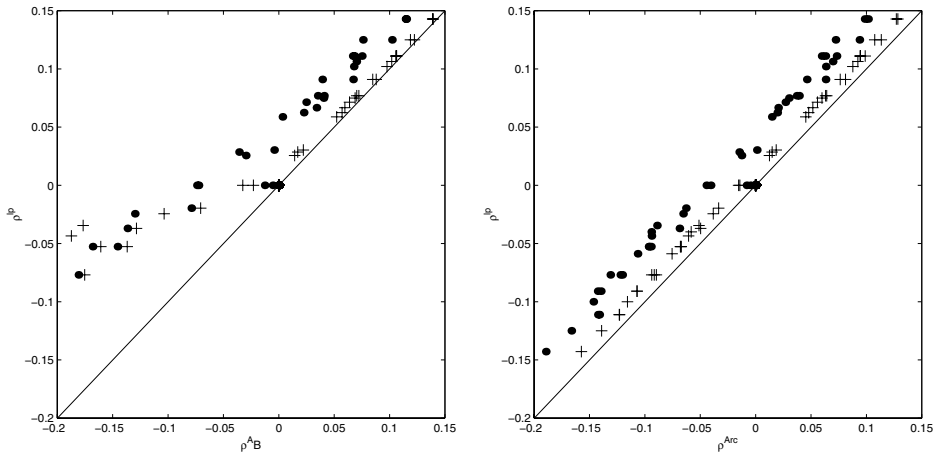


Figure 11.2 AdaBoost's and Arc-GV's margins (on the x–axis) vs. the optimal margin ρ^{lp} (y–axis) for a toy problem (a dot after 100, a "+" after 1000 iterations).

whereas for Arc-GV the optimum is always found. Having a look at equations (11.8) and (11.11) we see that in AdaBoost $\|\boldsymbol{\alpha}\|_1$ will converge faster to ∞ than in Arc-GV (for $\rho^{\text{lp}} > 0$), supporting the third observation.
Moreover, the number of iterations necessary to converge to a *good* solution seems to be reasonable, but for an (almost) *optimal* solution the number of iterations is unexpectedly high. This implies that for real world hypothesis sets the number of iterations needed to find an almost optimal solution can become prohibitive, but we conjecture that in practice a reasonably good approximation to the optimum is provided by both, AdaBoost and Arc-GV.

11.3 ν-Algorithms

For the LP-AdaBoost approach it has been shown for noisy problems, that the generalization performance is usually not as good as the one of AdaBoost [Grove and Schuurmans, 1998, Breiman, 1999, Rätsch, 1998]. From Theorem 1.15 this fact becomes clear, as the minimum of the right hand side of inequality (1.50) need not necessarily be achieved with a maximum margin. We now propose an algorithm where we can directly control the number of margin errors and therefore also the contribution of both terms in inequality (1.50) separately (cf. Theorem 11.3). We first consider a small hypothesis class and end up with a linear program – ν-LP-AdaBoost. In Subsection 11.3.2 we then combine this algorithm with the ideas from Section 11.2 and get two algorithms – ν-Arc and RoBoost – which approximate the ν-LP solution.

11.3.1 ν-LP-AdaBoost

Let us consider the case where we have given a (finite) set $G = \{g : \mathbf{x} \mapsto [-1, 1]\}$ of T hypothesis. To find the coefficients $\boldsymbol{\alpha}$ for the combined hypothesis $f(\mathbf{x})$ we extend the LP-AdaBoost algorithm [Grove and Schuurmans, 1998, Rätsch, 1998, Rätsch et al., 1998] and solve the following linear optimization problem, similar in spirit to [Schölkopf et al., 1998c]:

$$\begin{array}{lrl} \text{maximize} & \rho - \frac{1}{\nu m}\sum_{i=1}^{m}\xi_i & \\ \text{subject to} & \rho(\mathbf{z}_i, \boldsymbol{\alpha}) \geq \rho - \xi_i & \text{for all } 1 \leq i \leq m \\ & \xi_i, \alpha_t, \rho \geq 0 & \text{for all } 1 \leq t \leq T \text{ and } 1 \leq i \leq m \\ & \|\boldsymbol{\alpha}\|_1 = 1 & \end{array} \tag{11.12}$$

This algorithm does not force all margins to be beyond zero and we get a *soft margin* classification with a regularization constant $\frac{1}{\nu m}$. The following proposition shows that ν has an immediate interpretation:

Proposition 11.2
Suppose we run the algorithm given in (11.12) on some data with the resulting optimal $\rho > 0$. Then

1. ν upper-bounds the fraction of margin errors.
2. $1 - \nu$ is an upper bound on the fraction of patterns with a margin larger than ρ.

For an outline of the proof see Figure 11.3 (cf. also Schölkopf et al. [1998c], Graepel et al. [1999]).

SVM connection

Interestingly, if we choose the hypothesis set G with $T = m$ hypotheses to be

$$g_i(\mathbf{x}) = y_i k(\mathbf{x}_i, \mathbf{x}), (i = 1 \ldots m) \ , \tag{11.13}$$

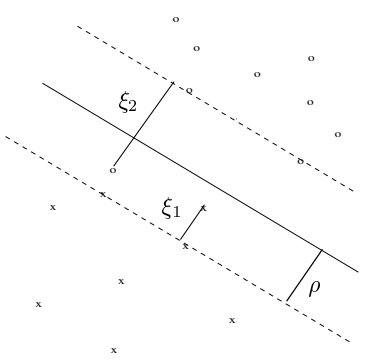


Figure 11.3 Graphical proof of the ν-property. Imagine decreasing ρ, starting from some large value. The first term in $\nu\rho - \frac{1}{m}\sum_{i=1}^{m}\xi_i$ (cf. (11.12)) will decrease proportionally to ν, while the second term will decrease proportionally to the fraction of points outside of the margin area. Hence, ρ will shrink as long as the latter fraction is larger than ν. At the optimum, it therefore must be $\leq \nu$ (Proposition 11.2, 1). Next, imagine increasing ρ, starting from 0. Again, the change in the first term is proportional to ν, but this time, the change in the second term is proportional to the fraction of patterns in the margin area or *exactly on* the margin. Hence, ρ will grow as long as the latter fraction is smaller than ν, eventually leading to Proposition 11.2, 2.

then the algorithm above is equivalent to the Linear Programming Machines (LPMs) [Graepel et al., 1999] with some kernel $k(\cdot,\cdot)$ (except for the additional constraints $\alpha_t \geq 0$). In ν-LP-AdaBoost we are not restricted to a fixed kernel type as we can use arbitrary base hypotheses (and also an arbitrary number of hypotheses – as long as we can fit the problem into a computer).

robustness

Since the slack variables ξ_i only enter the cost function linearly, their absolute size is not important. Loosely speaking, this is due to the fact that for the optimum of the primal objective function, only derivatives wrt. the primal variables matter, and the derivative of a linear function is constant.

In the case of SVMs, where the hypotheses can be thought as vectors in some feature space, this statement can be translated into a precise rule for distorting training patterns without changing the solution: we can move them locally parallel to the above vector. This yields a desirable *resistance* (or robustness) property. For general base hypotheses, it is difficult to state the allowed class of transformations in input space. Nevertheless, note that the algorithm essentially depends on the *number* of outliers, not on the size of the margin error [Schölkopf et al., 1998c].

11.3.2 ν-Arc and RoBoost

Suppose, we have a very large (but finite or at least with finite covering number $\mathcal{N}_\epsilon$) base hypothesis class G. Then it is very difficult to solve (11.12) as (11.5) directly. To this end, we propose two algorithms – ν-Arc and RoBoost – that can approximate the solution of (11.12).

rewriting ν-LP-AdaBoost

The optimal ρ for fixed margins $\rho(\mathbf{z}_i, \boldsymbol{\alpha})$ in (11.12) can be written as

$$\rho_\nu(\boldsymbol{\alpha}) := \operatorname*{argmax}_{\rho\in[0,1]} \left(\rho - \frac{1}{\nu m} \sum_{i=1}^{m} (\rho - \rho(\mathbf{z}_i, \boldsymbol{\alpha}))_+ \right). \tag{11.14}$$

where $(\xi)_+ := \max(\xi, 0)$. Setting $\xi_i := (\rho_\nu(\boldsymbol{\alpha}) - \rho(\mathbf{z}_i, \boldsymbol{\alpha}))_+$ and subtracting $\frac{1}{\nu m}\sum_{i=1}^{m} \xi_i$ from the resulting inequality on both sides, yields (for all $1 \leq i \leq m$)

$$\rho(\mathbf{z}_i, \boldsymbol{\alpha}) + \xi_i \geq \rho_\nu(\boldsymbol{\alpha}) \tag{11.15}$$

$$\rho(\mathbf{z}_i, \boldsymbol{\alpha}) + \xi_i - \frac{1}{\nu m} \sum_{i=1}^{m} \xi_i \geq \rho_\nu(\boldsymbol{\alpha}) - \frac{1}{\nu m} \sum_{i=1}^{m} \xi_i \ . \tag{11.16}$$

Two more substitutions are needed to transform the problem into one which can be solved by the AdaBoost algorithm, i.e., (11.6) and (11.7). In particular we have to get rid of the slack variables ξ_i again by absorbing them into quantities similar to $\rho(\mathbf{z}_i, \boldsymbol{\alpha})$ and $\rho(\boldsymbol{\alpha})$.

introducing the soft margin

This works as follows: on the right hand side of (11.16) we have the objective function (cf. (11.12)) and on the left hand side a term that depends nonlinearly on $\boldsymbol{\alpha}$. Defining

$$\tilde{\rho}_\nu(\boldsymbol{\alpha}) := \rho_\nu(\boldsymbol{\alpha}) - \frac{1}{\nu m} \sum_{i=1}^{m} \xi_i \tag{11.17}$$

$$\tilde{\rho}_\nu(\mathbf{z}_i, \boldsymbol{\alpha}) := \rho(\mathbf{z}_i, \boldsymbol{\alpha}) + \xi_i - \frac{1}{\nu m} \sum_{i=1}^{m} \xi_i, \tag{11.18}$$

which we substitute for $\rho(\boldsymbol{\alpha})$ and $\rho(\mathbf{z}, \boldsymbol{\alpha})$ in (11.5), respectively, we obtain a new optimization problem. Note that $\tilde{\rho}_\nu(\boldsymbol{\alpha})$ and $\tilde{\rho}_\nu(\mathbf{z}_i, \boldsymbol{\alpha})$ play the role of a *corrected* margin. We obtain a non-linear min-max problem

$$\begin{array}{lrl} \text{maximize} & \tilde{\rho}(\boldsymbol{\alpha}) & \\ \text{subject to} & \tilde{\rho}(\mathbf{z}_i, \boldsymbol{\alpha}) \geq \tilde{\rho}(\boldsymbol{\alpha}) & \text{for all } 1 \leq i \leq m \\ & \alpha_t \geq 0 & \text{for all } 1 \leq t \leq T \\ & \|\boldsymbol{\alpha}\|_1 = 1 & \end{array}, \tag{11.19}$$

which Arc-GV and AdaBoost can solve approximately (cf. Section 11.2). Hence, replacing the margin ρ by $\tilde{\rho}(\mathbf{z}, \boldsymbol{\alpha})$ in equations (11.4), (11.6) and (11.7), we obtain two new algorithms which we refer to as *ν-Arc* and *RoBoost*. Algorithm 11.1 contains the pseudocode of RoBoost. To get the ν-Arc algorithm, the two marked

Algorithm 11.1 : RoBoost

argument: `Training sample,` $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathcal{X}$, $Y = \{y_1, \ldots, y_m\} \subset \{\pm 1\}$
`Number of iterations,` T
returns: `Convex combination of functions from` G.

function $E = \tilde{G}(\boldsymbol{\alpha})$

$\rho_\nu := \operatorname{argmax}_{\rho \in [-1,1]} \rho - \frac{1}{\nu m} \sum_{i=1}^{m} (\rho(\mathbf{z}_i, \boldsymbol{\alpha}) - \rho)_+$

`Set` $\xi_i := (\rho(\mathbf{z}_i, \boldsymbol{\alpha}) - \rho_\nu)_+$ `for all` $i = 1, \ldots, m$

$*$ $E := \sum_{i=1}^{m} \exp\left[-\|\boldsymbol{\alpha}\|_1 \left(\rho(\mathbf{z}_i, \boldsymbol{\alpha}) + \xi_i - \frac{1}{\nu m} \sum_{i=1}^{m} \xi_i\right)\right]$

end

function `RoBoost`(X, Y, T, ν)

for all $i = 1, \ldots, m$

$w^1(\mathbf{z}_i) := 1/m$

endfor

for all t `from` $\{1, \ldots, T\}$

$g_t := L(X, Y, \mathbf{w}^t)$

$*$ $\alpha_t := \operatorname{argmax}_{\alpha_t \geq 0} \tilde{G}(\boldsymbol{\alpha}^t)$

for all i `from` $i = 1, \ldots, m$

$w^{t+1}(\mathbf{z}_i) := \exp\left[-\rho(\mathbf{z}_i, \boldsymbol{\alpha}^t) - (\rho(\mathbf{z}_i, \boldsymbol{\alpha}^t) - \rho_\nu^t)_+\right]$

endfor

$\mathbf{w}^{t+1} := \mathbf{w}^{t+1} / \|\mathbf{w}^{t+1}\|_1$

endfor

return $f = \frac{\sum_{t=1}^{T} \alpha_t g_t}{\sum_{i=1}^{T} \alpha_t}$.

end

L is a learning algorithm that chooses a classifier from G to minimize weighted training error. For `ν-Arc` only the lines marked with "$*$" have to be changed (see text).

lines in Algorithm 11.1 have to be replaced by

$$E := \sum_{i=1}^{m} \exp\left[-\|\boldsymbol{\alpha}\|_1 \left(\rho(\mathbf{z}_i, \boldsymbol{\alpha}) + \xi_i - \frac{1}{\nu m}\sum_{i=1}^{m} \xi_j - \tilde{\rho}_\nu(\boldsymbol{\alpha}^{t-1})\right)\right],$$

where $\xi_i \equiv (\rho(\mathbf{z}_i, \boldsymbol{\alpha}) - \rho_\nu)_+$ and $\alpha_t := \operatorname{argmax}_{\alpha_t \in [0,1]} \tilde{G}(\boldsymbol{\alpha}^t)$.

computational complexity

Clearly, the computational costs for determining the hypothesis weight α_t is considerably higher than for AdaBoost. The implementation given in Algorithm 11.1 is $\mathcal{O}(m^2 \log^2(m))$ of simple operations (maybe there exist better implementations). This might be a problem, if the base hypothesis is very simple and computing α_t takes a relatively high fraction of the computing time. To avoid this problem, one can use similar approximation approaches as used in Schapire and Singer [1998].

the ν bound

We can now state interesting properties for RoBoost and ν-Arc by using Theorem 1.15 that bounds the generalization error $R(f)$ for ensemble methods. In our case $R_\rho(f) \leq \nu$ by construction (i.e., the number of patterns with a margin smaller than ρ, cf. Proposition 11.2), thus we get the following simple reformulation of this bound:

Theorem 11.3

Let $p(\mathbf{x}, y)$ be a distribution over $\mathcal{X} \times [-1, 1]$, and let X be a sample of m examples chosen iid according to p. Suppose the base-hypothesis space G has VC dimension h, and let $\delta > 0$. Then with probability at least $1 - \delta$ over the random choice of the training set X, Y, every function $f \in \operatorname{co}(G)$ generated by the algorithms above satisfies the following bound for all $\nu \in (0, 1)$ with $\rho_\nu > 0$.

$$R(f) \leq \nu + \sqrt{\frac{c}{m}\left(\frac{h \log^2(m/h)}{\rho_\nu^2} + \log\left(\frac{1}{\delta}\right)\right)} \tag{11.20}$$

The tradeoff in minimizing the right hand side between the first and the second term is controlled directly by an easy interpretable regularization parameter ν.

11.4 Experiments

In a first study, we show a set of toy experiments to illustrate the general behavior of Arc-GV and RoBoost. As base hypothesis class G we use RBF networks [Rätsch et al., 1998], and as data a two-class problem generated from several 2D Gauss blobs (cf. footnote 3 on page 212) where we randomly flipped $\sigma = 0\%, 10\%, 20\%, 25\%$ of the labels. We obtained the following results:

- ν-Arc and RoBoost lead to approximately νm patterns that are effectively used in the training of the base learner: Figure 11.4 (upper) shows the fraction of patterns that have high average weights during the learning process (i.e., $\sum_{t=1}^{T} w_t(\mathbf{z}_i) > 1/2m$). We find that the number of the latter increases (almost) linearly with ν. This follows from (11.17) as the (soft) margin of patterns with $\rho(\mathbf{z}, \boldsymbol{\alpha}) < \rho_\nu$ is set to ρ_ν and therefore the weight of those patterns will be the same.

- Both algorithms lead to the fraction ν of margin errors (cf. dashed line in Figure 11.4) exactly as predicted in Proposition 11.2.
- The (estimated) test error, averaged over 10 training sets, exhibits a rather flat minimum in ν (figure 11.4 (lower)). This indicates that just as for ν-SVMs, where corresponding results have been obtained, ν is a well-behaved parameter in the sense that a slight misadjustment it is not harmful.
- The ν algorithms are more robust against label noise than AdaBoost and Arc-GV (which we recover for $\nu = 0$). As illustrated in Figure 11.5, also for increasing noise σ the minimum around the optimal ν stays reasonably flat. This coincides with the interpretation of Theorem 11.3 that an optimal ν should increase with the noise level.
- The results of ν-Arc and RoBoost are almost the same. This coincides with the observations in Section 11.2 for $\rho > 0$.
- Finally, a good parameter of ν can already be inferred from prior knowledge of the expected error. Setting it to a value similar to the latter provides a good starting point for further optimization (cf. Theorem 11.3).

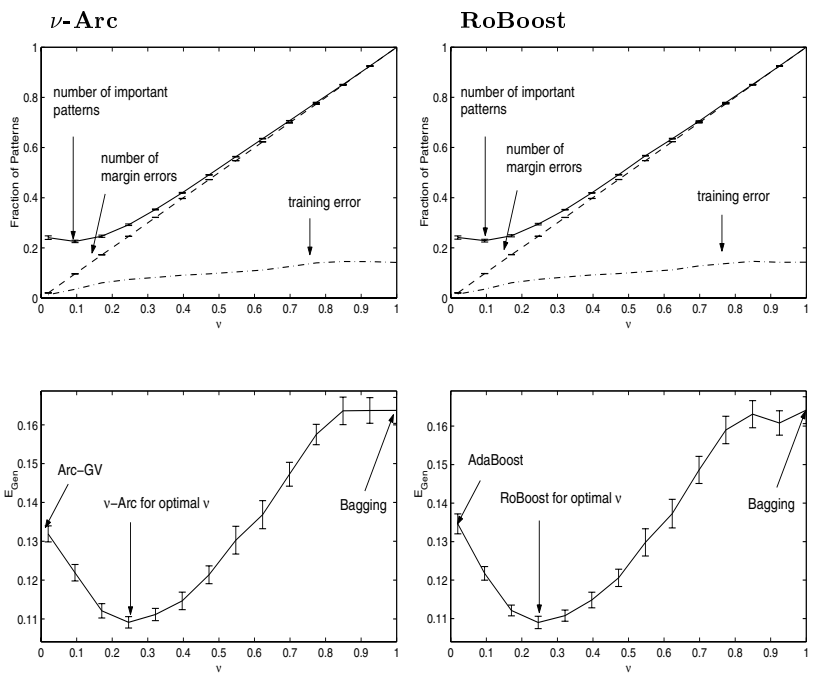


Figure 11.4 Toy experiment ($\sigma = 0$): the upper shows the average fraction of *important* patterns, the av. fraction of margin errors and the av. training error for different values of the regularization constant ν for ν-Arc/RoBoost. The bottom plots show the corresponding generalization error. In both cases the parameter ν allows us to reduce the test errors to values much lower than for the hard margin algorithm (for $\nu = 0$ we recover Arc-GV/AdaBoost and for $\nu = 1$ we get Bagging.)

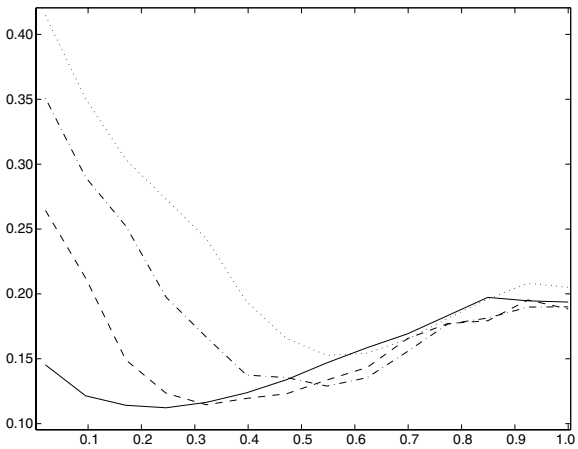


Figure 11.5 Illustration of RoBoost's robustness. Depicted is RoBoost's generalization error over ν for different label noise levels (on the training set; solid=0%, dashed=10%, dot-dashed=20%, and dotted=25%). Also, the minimal generalization error is better than for plain AdaBoost, which achieved 12.3, 13.8, 15.8 and 17.7 respectively. Especially for higher noise levels RoBoost performed relatively better (best results 11.2%, 11.5%, 12.9%, and 15.3% respectively).

Note that for $\nu = 1$ we recover the Bagging algorithm (if we used bootstrap samples), as the weights of all patterns will be the same ($w_t(\mathbf{z}_i) = 1/m$ for all $i = 1, \ldots, m$) and also the hypothesis weights will be constant ($\alpha_t \sim 1/T$ for all $t = 1, \ldots, T$) . This can be seen by setting $\rho_\nu = 1$ and α_t to an arbitrary positive constant (as $\mathcal{G}(\boldsymbol{\alpha})$ will not depend on α_t in this case).

Finally, we present a small comparison on ten benchmark data sets obtained from the UCI benchmark repository [Merz and Murphy, 1998]. We analyze the performance of single RBF networks, AdaBoost, ν-Arc and RBF-SVMs (we assume that ν-Arc and RoBoost perform very similar). For AdaBoost and ν-Arc we use RBF networks [Rätsch et al., 1998] as base hypothesis. The model parameters of RBF (number of centers etc.), ν-Arc (ν) and SVMs (σ, C) are optimized using 5-fold cross-validation. More details on the experimental setup can be found in [Rätsch et al., 1998]. Figure 11.1 shows the generalization error estimates (after averaging over 100 realizations of the data sets) and the confidence interval. The results of best classifier and the classifiers that are not significantly worse are set in bold face. To test the significance, we used a t-test ($p = 80\%$). On eight out of the ten data sets, ν-Arc performs significantly better than AdaBoost. This clearly shows the superior performance of ν-Arc and supports this soft margin approach for AdaBoost. Furthermore, we find that the performances of ν-Arc and the SVM are comparable. In three cases the SVM performs better and in two cases ν-Arc performs best. Summarizing, AdaBoost is useful for low noise cases, where the classes are separable. ν-Arc/RoBoost extends the applicability of boosting to problems that are difficult to separate and should be applied if the data are noisy.

	RBF	AB	ν-Arc	SVM
Banana	10.8±0.06	12.3±0.07	**10.6±0.05**	11.5±0.07
B.Cancer	27.6±0.47	30.4±0.47	**25.8±0.46**	**26.0±0.47**
Diabetes	24.3±0.19	26.5±0.23	**23.7±0.20**	**23.5±0.17**
German	24.7±0.24	27.5±0.25	24.4±0.22	**23.6±0.21**
Heart	17.6±0.33	20.3±0.34	**16.5±0.36**	**16.0±0.33**
Ringnorm	**1.7±0.02**	1.9±0.03	**1.7±0.02**	**1.7±0.01**
F.Sonar	34.4±0.20	35.7±0.18	34.4±0.19	**32.4±0.18**
Thyroid	**4.5±0.21**	**4.4±0.22**	**4.4±0.22**	4.8±0.22
Titanic	23.3±0.13	**22.6±0.12**	23.0±0.14	**22.4±0.10**
Waveform	10.7±0.11	10.8±0.06	**10.0±0.07**	**9.9±0.04**

Table 11.1 Generalization error estimates and confidence intervals. The best classifiers for a particular data set are marked in bold face (see text).

11.5 Conclusion

We analyzed the AdaBoost algorithm and found that Arc-GV and AdaBoost are suitable for approximating the solution of non-linear min-max problems over huge hypothesis classes. We re-parameterized the LP_{Reg}-AdaBoost algorithm (cf. Grove and Schuurmans [1998], Rätsch et al. [1998], Rätsch [1998]) and introduced a new regularization constant ν that controls the fraction of patterns inside the margin area. The new parameter is highly intuitive and has to be optimized only on a fixed interval $[0, 1]$.

Using the fact that AdaBoost can approximately solve min-max problems, we found a formulation of AdaBoost $-$RoBoost$-$ and similarly of Arc-GV $-\nu$-Arc$-$ that implements the ν-idea for Boosting by defining an appropriate *soft margin* (cf. also Bennett and Mangasarian [1992]). The present paper extends previous work on regularizing boosting (DOOM [Mason et al., 1999], $\mathrm{AdaBoost}_{\mathrm{Reg}}$ [Rätsch et al., 1998]) and shows the utility and flexibility of the soft margin approach for AdaBoost.

We found empirically that the generalization performance in RoBoost and ν-Arc depends only slightly on the optimal choice of the regularization constant. This makes model selection (e.g., via cross-validation) much easier.

Future work will study the detailed regularization properties of the regularized versions of AdaBoost, in particular how it compares to ν-LP Support Vector Machines.

Acknowledgments

Partial funding from EC STORM project grant (25387) and DFG grant (Ja 379/52,71) is gratefully acknowledged. This work was done while AS and BS were at GMD FIRST.

12 Functional Gradient Techniques for Combining Hypotheses

Llew Mason
Research School of Information Sciences and Engineering
Australian National University
Canberra, ACT, 0200, Australia
lmason@syseng.anu.edu.au

Jonathan Baxter
Research School of Information Sciences and Engineering
Australian National University
Canberra, ACT, 0200, Australia
Jon.Baxter@anu.edu.au

Peter L. Bartlett
Research School of Information Sciences and Engineering
Australian National University
Canberra, ACT, 0200, Australia
Peter.Bartlett@anu.edu.au

Marcus Frean
Department of Computer Science and Electrical Engineering
University of Queensland
Brisbane, QLD, 4072, Australia
marcusf@elec.uq.edu.au

Much recent attention, both experimental and theoretical, has been focussed on classification algorithms which produce voted combinations of classifiers. Recent theoretical work has shown that the impressive generalization performance of algorithms like AdaBoost can be attributed to the classifier having large margins on the training data.

We present abstract algorithms for finding linear and convex combinations of functions that minimize arbitrary cost functionals (i.e., functionals that do not necessarily depend on the margin). Many existing voting methods can be shown to

be special cases of these abstract algorithms. Then, following previous theoretical results bounding the generalization performance of convex combinations of classifiers in terms of general cost functions of the margin, we present a new algorithm, DOOM II, for performing a gradient descent optimization of such cost functions.

Experiments on several data sets from the UC Irvine repository demonstrate that DOOM II generally outperforms AdaBoost, especially in high noise situations. Margin distribution plots verify that DOOM II is willing to "give up" on examples that are too hard in order to avoid overfitting. We also show that the overfitting behavior exhibited by AdaBoost can be quantified in terms of our proposed cost function.

12.1 Introduction

There has been considerable interest recently in *voting methods* for pattern classification, which predict the label of a particular example using a weighted vote over a set of base classifiers. For example, AdaBoost [Freund and Schapire, 1997] and Bagging [Breiman, 1996] have been found to give significant performance improvements over algorithms for the corresponding base classifiers [Drucker and Cortes, 1996, Freund and Schapire, 1996, Quinlan, 1996a, Dietterich, 1998, Schwenk and Bengio, 1998, Bauer and Kohavi, 1997, Maclin and Opitz, 1997], and have led to the study of many related algorithms [Breiman, 1999, Schapire and Singer, 1998, Friedman et al., 1998, Rätsch et al., 1998, Duffy and Helmbold, 1999, Friedman, 1999]. Recent theoretical results suggest that the effectiveness of these algorithms is due to their tendency to produce *large margin classifiers*. (See Section 1.4 for a definition of margins and a review of these results.)

Mason, Bartlett, and Baxter [1999] presented improved upper bounds on the misclassification probability of a combined classifier in terms of the average over the training data of a certain *cost function* of the margins. That paper also described experiments with an algorithm, DOOM, that modifies the classifier weights of an *existing* combined classifier in order to minimize this cost function. This algorithm exhibits performance improvements over AdaBoost, which suggests that these margin cost functions are appropriate quantities to optimize. Unlike the DOOM algorithm (which does not provide a method for choosing the base classifiers), the DOOM II algorithm presented in this chapter provides an iterative method for choosing both the base classifiers *and* their weights so as to minimize the cost functions suggested by the theoretical analysis of [Mason et al., 1999].

In this chapter, we present a general algorithm, MarginBoost, for choosing a combination of classifiers to optimize the sample average of any cost function of the margin. MarginBoost performs gradient descent in function space, at each iteration choosing a base classifier to include in the combination so as to maximally reduce the cost function. The idea of performing gradient descent in function space in this way is due to Breiman [1999]. It turns out that, as in AdaBoost, the choice of the base classifier corresponds to a minimization problem involving weighted

classification error. That is, for a certain weighting of the training data, the base classifier learning algorithm attempts to return a classifier that minimizes the weight of misclassified training examples.

There is a simpler and more abstract way to view the MarginBoost algorithm. In Section 12.2, we describe a class of algorithms (called AnyBoost) which are gradient descent algorithms for choosing linear combinations of elements of an inner product space so as to minimize some cost functional. Each component of the linear combination is chosen to maximize a certain inner product. (In MarginBoost, this inner product corresponds to the weighted training error of the base classifier.) In Section 12.5, we give convergence results for this class of algorithms. For MarginBoost with a convex cost function, these results show that, with a particular choice of the step-size, if the base classifier minimizes the appropriate weighted error then the algorithm converges to the global minimum of the cost function.

In Section 12.3, we show that this general class of algorithms includes as special cases a number of popular and successful voting methods, including AdaBoost [Freund and Schapire, 1997], an extension of AdaBoost to combinations of real-valued functions [Schapire and Singer, 1998], and LogitBoost [Friedman et al., 1998]. That is, all of these algorithms implicitly minimize some margin cost function by gradient descent.

In Section 12.4, we review the theoretical results from [Mason et al., 1999] bounding the error of a combination of classifiers in terms of the sample average of certain cost functions of the margin. The cost functions suggested by these results are significantly different from the cost functions that are implicitly minimized by the methods described in Section 12.3. In Section 12.6, we present experimental results for the MarginBoost algorithm with cost functions that are motivated by the theoretical results. These experiments show that the new algorithm typically outperforms AdaBoost, and that this is especially true with label noise. In addition, the theoretically-motivated cost functions provide good estimates of the error of AdaBoost, in the sense that they can be used to predict its overfitting behaviour.

Similar techniques for directly optimizing margins (and related quantities) have been described by several authors. Rätsch et al. [1998] show that versions of AdaBoost modified to use regularization are more robust for noisy data. Friedman [1999] describes general "boosting" algorithms for regression and classification using various cost functions and presents specific cases for boosting decision trees. Duffy and Helmbold [1999] describe two algorithms (GeoLev and GeoArc) which attempt to produce combined classifiers with *uniformly* large margins on the training data. Freund [1999] presents a new boosting algorithm which uses example weights similar to those suggested by the theoretical results from [Mason et al., 1999].

12.2 Optimizing Cost Functions of the Margin

We begin with some notation. We assume that examples $(\mathbf{x}, y)$ are randomly generated according to some unknown probability distribution $\mathcal{D}$ on $X \times Y$ where X is the space of measurements (typically $X \subseteq \mathbb{R}^N$) and Y is the space of labels (Y is usually a discrete set or some subset of $\mathbb{R}$).

Although the abstract algorithms of the following section apply to many different machine learning settings, our primary interest in this chapter is voted combinations of classifiers of the form $\operatorname{sgn}(F(\mathbf{x}))$, where

$$F(\mathbf{x}) = \sum_{t=1}^{T} w_t f_t(\mathbf{x}),$$

$f_t : X \to \{\pm 1\}$ are base classifiers from some fixed class $\mathcal{F}$ and $w_t \in \mathbb{R}$ are the classifier weights. Recall (Definition 1.2) that the *margin* of an example $(\mathbf{x}, y)$ with respect to the classifier $\operatorname{sgn}(F(\mathbf{x}))$ is defined as $yF(\mathbf{x})$.

Given a set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ of m labelled examples generated according to $\mathcal{D}$ we wish to construct a voted combination of classifiers of the form described above so that $P_{\mathcal{D}}(\operatorname{sgn}(F(\mathbf{x})) \neq y)$ is small. That is, the probability that F incorrectly classifies a random example is small. Since $\mathcal{D}$ is unknown and we are only given a training set S, we take the approach of finding voted classifiers which minimize the sample average of some cost function of the margin. That is, for a training set S we want to find F such that

$$C(F) = \frac{1}{m} \sum_{i=1}^{m} C(y_i F(\mathbf{x}_i)) \tag{12.1}$$

is minimized for some suitable cost function $C : \mathbb{R} \to \mathbb{R}$. Note that we are using the symbol C to denote both the cost function of the real margin $yF(\mathbf{x})$, and the cost *functional* of the function F. Which interpretation is meant should always be clear from the context.

12.2.1 AnyBoost

One way to produce a weighted combination of classifiers which optimizes (12.1) is by gradient descent in function space, an idea first proposed by Breiman [1999]. Here we present a more abstract treatment that shows how many existing voting methods may be viewed as gradient descent in a suitable *inner product* space.

At an abstract level we can view the base hypotheses $f \in \mathcal{F}$ and their combinations F as elements of an inner product space $(\mathcal{X}, \langle , \rangle)$. In this case, $\mathcal{X}$ is a linear space of functions that contains $\operatorname{lin}(\mathcal{F})$, the set of all linear combinations of functions in $\mathcal{F}$, and the inner product is defined by

$$\langle F, G \rangle := \frac{1}{m} \sum_{i=1}^{m} F(\mathbf{x}_i) G(\mathbf{x}_i) \tag{12.2}$$

for all $F, G \in \text{lin}(\mathcal{F})$. However, the AnyBoost algorithms defined in this section and their convergence properties studied in Section 12.5 are valid for any cost function and inner product. For example, they will hold in the case $\langle F, G \rangle := \int_X F(\mathbf{x})G(\mathbf{x})dP(\mathbf{x})$ where P is the marginal distribution on the input space generated by $\mathcal{D}$.

Now suppose we have a function $F \in \text{lin}(\mathcal{F})$ and we wish to find a new $f \in \mathcal{F}$ to add to F so that the cost $C(F + \epsilon f)$ decreases, for some small value of ϵ. Viewed in function space terms, we are asking for the "direction" f such that $C(F + \epsilon f)$ most rapidly decreases. Viewing the cost C as a *functional* on $\text{lin}(\mathcal{F})$, the desired direction is simply $-\nabla C(F)(\mathbf{x})$, the negative of the functional derivative of C at F. Here, $\nabla C(F)$ is the unique function such that for any $f \in \mathcal{X}$,

$$C(F + f) = C(F) + \langle \nabla C(F), f \rangle + o(\|f\|). \tag{12.3}$$

If we assume that C is differentiable everywhere then

$$\nabla C(F)(\mathbf{x}) := \left. \frac{\partial C(F + \alpha 1_{\mathbf{x}})}{\partial \alpha} \right|_{\alpha=0}, \tag{12.4}$$

where $1_{\mathbf{x}}$ if the indicator function of $\mathbf{x}$. Since we are restricted to choosing our new function f from $\mathcal{F}$, in general it will not be possible to choose $f = -\nabla C(F)$, so instead we search for an f with greatest inner product with $-\nabla C(F)$. That is, we should choose f to maximize

$$-\langle \nabla C(F), f \rangle .$$

This can be motivated by observing that (12.3) implies that, to first order in ϵ,

$$C(F + \epsilon f) = C(F) + \epsilon \langle \nabla C(F), f \rangle$$

and hence the greatest reduction in cost will occur for the f which maximizes $-\langle \nabla C(F), f \rangle$.

The preceding discussion motivates Algorithm 12.1, an iterative algorithm for finding linear combinations F of base hypotheses in $\mathcal{F}$ that minimize the cost $C(F)$. Note that we have allowed the base hypotheses to take values in an arbitrary set Y, we have not restricted the form of the cost or the inner product, and we have not specified what the step-sizes should be. Appropriate choices for these things will be made when we apply the algorithm to more concrete situations. Note also that the algorithm terminates when $-\langle \nabla C(F_t), f_{t+1} \rangle \leq 0$, i.e., when the weak learner $\mathcal{L}$ returns a base hypothesis f_{t+1} which *no longer points in the downhill direction* of the cost function $C(F)$. Thus, the algorithm terminates when, to first order, a step in function space in the direction of the base hypothesis returned by $\mathcal{L}$ would increase the cost.

Algorithm 12.1 : AnyBoost

Require :

- An inner product space $(\mathcal{X}, \langle , \rangle)$ containing functions mapping from X to some set Y.
- A class of base classifiers $\mathcal{F} \subseteq \mathcal{X}$.
- A differentiable cost functional $C: \mathrm{lin}\,(\mathcal{F}) \to \mathbb{R}$.
- A weak learner $\mathcal{L}(F)$ that accepts $F \in \mathrm{lin}\,(\mathcal{F})$ and returns $f \in \mathcal{F}$ with a large value of $-\langle \nabla C(F), f \rangle$.

Let $F_0(\mathbf{x}) := 0$.
for $t := 0$ to T **do**
  Let $f_{t+1} := \mathcal{L}(F_t)$.
  if $-\langle \nabla C(F_t), f_{t+1} \rangle \leq 0$ **then**
    return F_t.
  end if
  Choose w_{t+1}.
  Let $F_{t+1} := F_t + w_{t+1} f_{t+1}$
end for
return F_{T+1}.

12.2.2 AnyBoost.L_1

The AnyBoost algorithm can return an arbitrary linear combination of elements of the base hypothesis class. Such flexibility has the potential to cause overfitting. Indeed, Theorem 12.2 in the following section provides guaranteed generalization performance for certain classes of cost functions, provided the algorithm returns elements of $\mathrm{co}\,(\mathcal{F})$, that is *convex* combinations of elements from the base hypothesis class.[1] This consideration motivates Algorithm 12.2—AnyBoost.L_1—a normalized version of AnyBoost that only returns functions in the convex hull of the base hypothesis class $\mathcal{F}$.

Notice that the stopping criterion of AnyBoost.L_1 is $-\langle \nabla C(F_t), f_{t+1} - F_t \rangle \leq 0$, rather than $-\langle \nabla C(F_t), f_{t+1} \rangle \leq 0$. To see why, notice that at every iteration F_t must lie in $\mathrm{co}\,(\mathcal{F})$. Hence, in incorporating a new component f_{t+1}, we update F_t to $(1-\alpha)F_t + \alpha f_{t+1}$ for some $\alpha \in [0,1]$. Hence, $F_{t+1} = F_t + \alpha(f_{t+1} - F_t)$ which corresponds to stepping in the direction corresponding to $f_{t+1} - F_t$. Geometrically, $-\langle \nabla C(F_t), f_{t+1} - F_t \rangle \leq 0$ implies that the change $F_{t+1} - F_t$ associated with the addition of f_{t+1} is not within 90° of $-\nabla C(F_t)$.

1. For convenience, we assume that the class $\mathcal{F}$ contains the zero function, or equivalently, that $\mathrm{co}\,(\mathcal{F})$ denotes the convex cone containing convex combinations of functions from $\mathcal{F}$ and the zero function.

Algorithm 12.2 : AnyBoost.L_1

Require :

- An inner product space $(\mathcal{X}, \langle , \rangle)$ containing functions mapping from X to some set Y.
- A class of base classifiers $\mathcal{F} \subseteq \mathcal{X}$.
- A differentiable cost functional $C : \mathrm{co}\,(\mathcal{F}) \to \mathbb{R}$.
- A weak learner $\mathcal{L}(F)$ that accepts $F \in \mathrm{co}\,(\mathcal{F})$ and returns $f \in \mathcal{F}$ with a large value of $-\langle \nabla C(F), f - F \rangle$.

Let $F_0(\mathbf{x}) := 0$.
for $t := 0$ to T **do**
  Let $f_{t+1} := \mathcal{L}(F_t)$.
  if $-\langle \nabla C(F_t), f_{t+1} - F_t \rangle \leq 0$ **then**
    return F_t.
  end if
  Choose w_{t+1}.
  Let $F_{t+1} := \dfrac{F_t + w_{t+1} f_{t+1}}{\sum_{s=1}^{t+1} |w_s|}$.
end for
return F_{T+1}.

12.2.3 AnyBoost.L_2

AnyBoost.L_1 enforces an L_1 constraint on the size of the combined hypotheses returned by the algorithm. Although for certain classes of cost functionals we have theoretical guarantees on the generalization performance of such algorithms (see Section 12.4), from an aesthetic perspective an L_2 constraint is more natural in an inner product space setting. In particular, we can then ask our algorithm to perform gradient descent on a regularized cost functional of the form

$$C(F) + \lambda \|F\|^2,$$

where λ is a regularization parameter, without needing to refer to the individual weights in the combination F (contrast with AnyBoost.L_1). In future work we plan to investigate the experimental performance of algorithms based on L_2 constraints.

With an L_2 rather than L_1 constraint, we also have the freedom to allow the weak learner to return general linear combinations in the base hypothesis class, not just single hypotheses.[2] In general a linear combination $F \in \mathrm{lin}\,(\mathcal{F})$ will be closer to the negative gradient direction than any single base hypothesis, hence stepping

2. The optimal direction in which to move for AnyBoost.L_1 is always a *pure* direction $f \in \mathcal{F}$ if the current combined hypothesis F_t is already on the convex hull of $\mathcal{F}$. So a weak learner that produces linear combinations will be no more powerful than a weak learner returning a single hypothesis in the L_1 case. This is not true for the L_2 case.

in the direction of F should lead to a greater reduction in the cost function, while still ensuring the overall hypothesis constructed is an element of $\text{lin}(\mathcal{F})$.

A weak learner $\mathcal{L}$ that accepts a direction G and attempts to choose an $f \in \mathcal{F}$ maximizing $\langle G, f \rangle$ can easily be converted to a weak learner $\mathcal{L}'$ that attempts to choose an $H \in \text{lin}(\mathcal{F})$ maximizing $\langle G, H \rangle$; the details are given in Algorithm 12.3. $\mathcal{L}'$ would then be substituted for $\mathcal{L}$ in the AnyBoost algorithm.

Algorithm 12.3 : $\mathcal{L}'$: a weak learner returning linear combinations

Require :

- An inner product space $(\mathcal{X}, \langle , \rangle)$ (with associated norm $\|F\|^2 := \langle F, F \rangle$) containing functions mapping from X to some set Y.
- A class of base classifiers $\mathcal{F} \subseteq \mathcal{X}$.
- A differentiable cost functional $C: \text{lin}(\mathcal{F}) \to \mathbb{R}$.
- A weak learner $\mathcal{L}(G)$ that accepts a "direction" $G \in S$ and returns $f \in \mathcal{F}$ with a large value of $\langle G, f \rangle$.
- A starting function $F_t \in \text{lin}(\mathcal{F})$.

Let $G_0 := -\nabla C(F_t)/\|\nabla C(F_t)\|$.
Let $H_0 := 0$.
for $t := 0$ to T **do**
  Let $h_{t+1} := \mathcal{L}(G_t)$.
  Let $H_{t+1} := \alpha H_t + \beta h_{t+1}$, with the constraints $\|H_{t+1}\| = 1$ and $\langle H_{t+1}, G_t \rangle$ maximal.
  if $\beta = 0$ **then**
    return H_t.
  end if
  Let $G_{t+1} := G_0 - H_{t+1}$.
end for
return H_{T+1}.

12.2.4 AnyBoost and Margin Cost Functionals

Since the main aim of this chapter is optimization of margin cost functionals, in this section we specialize the AnyBoost and AnyBoost.L_1 algorithms of the previous two sections by restricting our attention to the inner product (12.2), the cost (12.1), and $Y = \{\pm 1\}$. In this case,

$$\nabla C(F)(\mathbf{x}) = \{\, 0 \quad \text{if } \mathbf{x} \neq \mathbf{x}_i, i = 1 \ldots m \frac{1}{m} y_i C'(y_i F(\mathbf{x}_i)) \quad \text{if } \mathbf{x} = \mathbf{x}_i,$$

where $C'(z)$ is the derivative of the margin cost function with respect to z. Hence,

$$-\langle \nabla C(F), f \rangle = -\frac{1}{m^2} \sum_{i=1}^{m} y_i f(\mathbf{x}_i) C'(y_i F(\mathbf{x}_i)).$$

Any sensible cost function of the margin will be monotonically decreasing, hence $-C'(y_i F(\mathbf{x}_i))$ will always be positive. Dividing through by $-\frac{1}{m^2}\sum_{i=1}^m C'(y_i F(\mathbf{x}_i))$, we see that finding an f maximizing $-\langle \nabla C(F), f\rangle$ is equivalent to finding an f minimizing

$$-\sum_{i=1}^m y_i f(\mathbf{x}_i)\frac{C'(y_i F(\mathbf{x}_i))}{\sum_{i=1}^m C'(y_i F(\mathbf{x}_i))}. \tag{12.5}$$

Since $Y = \{\pm 1\}$, $y_i f(\mathbf{x}_i)$ is either 1 if $f(\mathbf{x}_i) = y_i$ or -1 if $f(\mathbf{x}_i) \neq y_i$. Hence (12.5) can be rewritten as

$$\sum_{i:f(\mathbf{x}_i)\neq y_i} D(i) - \sum_{i:f(\mathbf{x}_i)=y_i} D(i) = 2\sum_{i:f(\mathbf{x}_i)\neq y_i} D(i) - 1,$$

where $D(1),\ldots,D(m)$ is the distribution

$$D(i) := \frac{C'(y_i F(\mathbf{x}_i))}{\sum_{i=1}^m C'(y_i F(\mathbf{x}_i))}.$$

So finding an f maximizing $-\langle \nabla C(F), f\rangle$ is equivalent to finding f minimizing the weighted error

$$\sum_{i:f(\mathbf{x}_i)\neq y_i} D(i).$$

Making the appropriate substitutions in AnyBoost yields Algorithm 12.4, MarginBoost.

For AnyBoost.L_1 we require a weak learner that maximizes $-\langle \nabla C(F), f - F\rangle$ where F is the current convex combination. In the present setting this is equivalent to minimizing

$$\sum_{i=1}^m [F(\mathbf{x}_i) - f(\mathbf{x}_i)]\, y_i D(i)$$

with $D(i)$ as above. Making the appropriate substitutions in AnyBoost.L_1 yields Algorithm 12.5, MarginBoost.L_1.

12.3 A Gradient Descent View of Voting Methods

Many of the most successful voting methods are, for the appropriate choice of cost function and step-size, specific cases of the AnyBoost algorithm described above (or its derivatives).

The AdaBoost algorithm [Freund and Schapire, 1997] is arguably one of the most important developments in practical machine learning in the past decade. Many studies [Freund and Schapire, 1996, Quinlan, 1996a, Drucker and Cortes, 1996, Schwenk and Bengio, 1998] have demonstrated that AdaBoost can produce extremely accurate classifiers from base classifiers as simple as decision stumps or

Algorithm 12.4 : MarginBoost

Require :

- A differentiable cost function $C: \mathbb{R} \to \mathbb{R}$.
- A class of base classifiers $\mathcal{F}$ containing functions $f: X \to \{\pm 1\}$.
- A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ with each $(\mathbf{x}_i, y_i) \in X \times \{\pm 1\}$.
- A weak learner $\mathcal{L}(S, D)$ that accepts a training set S and a distribution D on the training set and returns base classifiers $f \in \mathcal{F}$ with small weighted error $\sum_{i: f(\mathbf{x}_i) \neq y_i} D(i)$.

Let $D_0(i) := 1/m$ for $i = 1, \ldots, m$.
Let $F_0(\mathbf{x}) := 0$.
for $t := 0$ to T **do**
  Let $f_{t+1} := \mathcal{L}(S, D_t)$.
  if $\sum_{i=1}^{m} D_t(i) y_i f_{t+1}(\mathbf{x}_i) \leq 0$ **then**
    return F_t.
  end if
  Choose w_{t+1}.
  Let $F_{t+1} := F_t + w_{t+1} f_{t+1}$

$$\text{Let } D_{t+1}(i) := \frac{C'\Big(y_i F_{t+1}(\mathbf{x}_i)\Big)}{\sum_{i=1}^{m} C'\Big(y_i F_{t+1}(\mathbf{x}_i)\Big)}$$

  for $i = 1, \ldots, m$.
end for
return F_{T+1}

as complex as neural networks or decision trees. The interpretation of AdaBoost as an algorithm which performs a gradient descent optimization of the sample average of a cost function of the margins has been examined by several authors [Breiman, 1999, Frean and Downs, 1998, Friedman et al., 1998, Duffy and Helmbold, 1999].

To see that the AdaBoost algorithm (shown in Table 1.2) is in fact MarginBoost using the cost function $C(\alpha) = e^{-\alpha}$ we need only verify that the distributions and stopping criteria are identical. The distribution D_{t+1} from AdaBoost can be rewritten as

$$\frac{\prod_{s=1}^{t} e^{-y_i w_s f_s(\mathbf{x}_i)}}{m \prod_{s=1}^{t} Z_s}. \tag{12.6}$$

Since D_{t+1} is a distribution then

$$m \prod_{s=1}^{t} Z_s = \sum_{i=1}^{m} \prod_{s=1}^{t} e^{-y_i w_s f_s(\mathbf{x}_i)} \tag{12.7}$$

and clearly

$$\prod_{s=1}^{t} e^{-y_i w_s f_s(\mathbf{x}_i)} = e^{-y_i F_t(\mathbf{x}_i)}. \tag{12.8}$$

Algorithm 12.5 : MarginBoost.L_1

Require :

- A differentiable cost function $C: \mathbb{R} \to \mathbb{R}$.
- A class of base classifiers $\mathcal{F}$ containing functions $f: X \to \{\pm 1\}$.
- A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ with each $(\mathbf{x}_i, y_i) \in X \times \{\pm 1\}$.
- A weak learner $\mathcal{L}(S, D, F)$ that accepts a training set S, a distribution D on the training set and a combined classifier F, and returns base classifiers $f \in \mathcal{F}$ with small weighted error: $\sum_{i=1}^{m} [F(\mathbf{x}_i) - f(\mathbf{x}_i)]\, y_i D(i)$.

Let $D_0(i) := 1/m$ for $i = 1, \ldots, m$.
Let $F_0(\mathbf{x}) := 0$.
for $t := 0$ to T **do**
  Let $f_{t+1} := \mathcal{L}(S, D_t, F_t)$.
  if $\sum_{i=1}^{m} D_t(i) y_i [f_{t+1}(\mathbf{x}_i) - F_t(\mathbf{x}_i)] \leq 0$ **then**
    return F_t.
  end if
  Choose w_{t+1}.

  Let $F_{t+1} := \dfrac{F_t + w_{t+1} f_{t+1}}{\sum_{s=1}^{t+1} |w_s|}$.

  Let $D_{t+1}(i) := \dfrac{C'\Big(y_i F_{t+1}(\mathbf{x}_i)\Big)}{\sum_{i=1}^{m} C'\Big(y_i F_{t+1}(\mathbf{x}_i)\Big)}$

  for $i = 1, \ldots, m$.
end for
return F_{T+1}

Substituting (12.7) and (12.8) into (12.6) gives the MarginBoost distribution for the cost function $C(\alpha) = e^{-\alpha}$. By definition of ϵ_t, the stopping criterion in AdaBoost is

$$\sum_{i: f_{t+1}(\mathbf{x}_i) \neq y_i} D_t(i) \geq \frac{1}{2}.$$

This is equivalent to

$$\sum_{i: f_{t+1}(\mathbf{x}_i) = y_i} D_t(i) - \sum_{i: f_{t+1}(\mathbf{x}_i) \neq y_i} D_t(i) \leq 0,$$

which is identical to the stopping criterion of MarginBoost.

Given that we have chosen f_{t+1} we wish to choose w_{t+1} to minimize

$$\sum_{i=1}^{m} C(y_i F_t(\mathbf{x}_i) + y_i w_{t+1} f_{t+1}(\mathbf{x}_i)).$$

Differentiating with respect to w_{t+1}, setting this to 0 and solving for w_{t+1} gives

$$w_{t+1} = \frac{1}{2} \ln \left(\frac{\sum_{i:f_{t+1}(\mathbf{x}_i)=y_i} D_t(i)}{\sum_{i:f_{t+1}(\mathbf{x}_i)\neq y_i} D_t(i)} \right).$$

This is exactly the setting of w_t used in the AdaBoost algorithm. So for this choice of cost function it is possible to find a closed form solution for the line search for optimal step-size at each round. Hence, AdaBoost is performing gradient descent on the cost functional

$$C(F) = \frac{1}{m} \sum_{i=1}^{m} e^{-y_i F(\mathbf{x}_i)}$$

with step-size chosen by a line search.

Schapire and Singer [1998] examine AdaBoost in the more general setting where classifiers can produce real values in $[-1, 1]$ indicating their confidence in $\{\pm 1\}$-valued classification. The general algorithm[3] they present is essentially AnyBoost with the cost function $C(yF(\mathbf{x})) = e^{-yF(\mathbf{x})}$ and base classifiers $f : X \to [-1, 1]$.

The ARC-X4 algorithm due to Breiman [1999] is approximately AnyBoost.L_1 with the cost function $C(\alpha) = (1 - \alpha)^5$ with a constant step-size.

Friedman et al. [1998] examine AdaBoost as an approximation to maximum likelihood. From this viewpoint they develop a more direct approximation (LogitBoost) which exhibits similar performance. LogitBoost is AnyBoost with the cost function $C(\alpha) = \log_2(1 + e^{-2\alpha})$ and step-size chosen via a single Newton-Raphson step.

Lee et al. [1996] describe an iterative algorithm for constructing convex combinations of basis functions to minimize a quadratic cost function. They use a constructive approximation result to prove the rate of convergence of this algorithm to the optimal convex combination. This algorithm can be viewed as gradient descent with a quadratic cost function $C(\alpha) = (1 - \alpha)^2$ and step-size decreasing at the rate $1/t$.

Table 12.1 summarizes the cost function and step-size choices for which AnyBoost and its derivatives approximately reduce to existing voting methods.

12.4 Theoretically Motivated Cost Functions

The following definition from [Mason et al., 1999] gives a condition on a cost function $C_N(\cdot)$ that suffices to prove upper bounds on error probability in terms of sample averages of $C_N(yF(\mathbf{x}))$. The condition requires the cost function $C_N(\alpha)$ to lie strictly above the mistake indicator function, $\operatorname{sgn}(-\alpha)$. How close $C_N(\alpha)$ can be to $\operatorname{sgn}(-\alpha)$ depends on a complexity parameter N.

3. They also present a base learning algorithm for decision trees which directly optimizes the exponential cost function of the margin at each iteration. This variant of boosting does not reduce to a gradient descent optimization.

Algorithm	Cost function	Step-size
AdaBoost [Freund and Schapire, 1996]	$e^{-yF(\mathbf{x})}$	Line search
ARC-X4 [Breiman, 1996]	$(1 - yF(\mathbf{x}))^5$	1
ConfidenceBoost [Schapire and Singer, 1998]	$e^{-yF(\mathbf{x})}$	Line search
LogitBoost [Friedman et al., 1998]	$\ln(1 + e^{-2yF(\mathbf{x})})$	Newton-Raphson
Constructive NN algorithm [Lee et al., 1996]	$(1 - yF(\mathbf{x}))^2$	$1/t$

Table 12.1 Summary of existing voting methods which can be viewed as gradient descent optimizers of margin cost functions.

Definition 12.1
A family $\{C_N : N \in \mathbb{N}\}$ of margin cost functions is *B-admissible* for $B \geq 0$ if for all $N \in \mathbb{N}$ there is an interval $I \subset \mathbb{R}$ of length no more than B and a function $\Psi_N : [-1, 1] \to I$ that satisfies

$$\operatorname{sgn}(-\alpha) \leq \mathbf{E}_Z(\Psi_N(Z)) \leq C_N(\alpha)$$

for all $\alpha \in [-1, 1]$, where $\mathbf{E}_Z(\cdot)$ denotes the expectation when Z is chosen randomly as $Z = (1/N)\sum_{i=1}^{N} Z_i$ with $Z_i \in \{\pm 1\}$ and $\Pr(Z_i = 1) = (1 + \alpha)/2$.

The following theorem from [Mason et al., 1999] gives a high probability upper bound on the generalization error of any convex combination of classifiers in terms of the sample average of $C_N(yF(\mathbf{x}))$ and a complexity term depending on N.

Theorem 12.2
For any B-admissible family $\{C_N : N \in \mathbb{N}\}$ of margin cost functions, any finite hypothesis class H and any distribution $\mathcal{D}$ on $X \times \{\pm 1\}$, with probability at least $1 - \delta$ over a random sample S of m labelled examples chosen according to $\mathcal{D}$, every N and every F in $\operatorname{co}(\mathcal{F})$ satisfies

$$\Pr[yF(\mathbf{x}) \leq 0] < \mathbf{E}_S[C_N(yF(\mathbf{x}))] + \epsilon_N,$$

where

$$\epsilon_N = \sqrt{\frac{B^2}{2m}(N \ln|\mathcal{F}| + \ln(N(N+1)/\delta))}.$$

A similar result applies for infinite classes $\mathcal{F}$ with finite VC-dimension.

In this theorem, as the complexity parameter N increases, the sample-based error estimate $\mathbf{E}_S[C_N(yF(\mathbf{x}))]$ decreases towards the training error (proportion of misclassified training examples). On the other hand, the complexity penalty term ϵ_N increases with N. Hence, in choosing the effective complexity N of the combined classifier, there is a trade-off between these two terms. Smaller cost functions give a more favourable trade-off. The left plot of Figure 12.1 illustrates a family $C_N(\cdot)$ of cost functions that satisfy the B-admissibility condition. Notice that these functions are significantly different from the exponential and logit cost functions that are

used in AdaBoost and LogitBoost respectively. Unlike the exponential and logit functions, $C_N(\alpha)$ is nonconvex and for large negative margins the value of $C_N(\alpha)$ is significantly smaller.

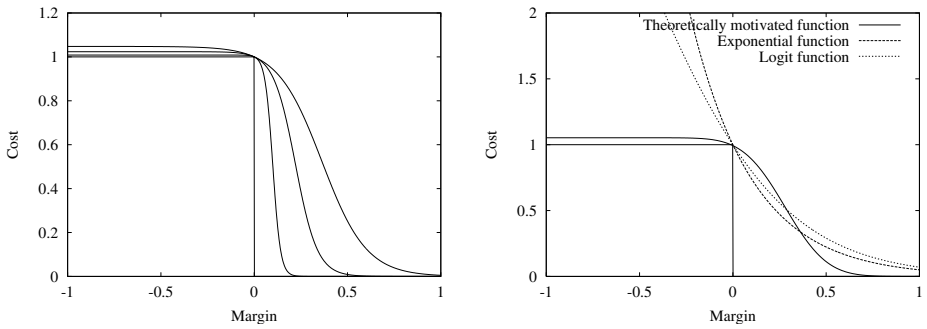


Figure 12.1 Cost functions $C_N(\alpha)$, for $N = 20, 50$ and 100, compared to the function $\operatorname{sgn}(-\alpha)$. Larger values of N correspond to closer approximations to $\operatorname{sgn}(-\alpha)$. The theoretically motivated cost function $C_{40}(\alpha)$ and the exponential and logit cost functions are also plotted together for comparison.

12.5 Convergence Results

In this section we prove convergence results for the abstract algorithms AnyBoost and AnyBoost.L_1, under quite weak conditions on the cost functional C. The prescriptions given for the step-sizes w_t in these results are for convergence guarantees only: in practice they will almost always be smaller than necessary, hence fixed small steps or some form of line search should be used.

Throughout this section we are interested in the limiting behaviour of AnyBoost (and its derivatives) and thus assume that the algorithms do not terminate after some fixed number of iterations T (although the algorithms can terminate due to internal termination conditions).

12.5.1 Convergence of AnyBoost

The following theorem supplies a specific step-size for AnyBoost and characterizes the limiting behaviour with this step-size.

Theorem 12.3
Let $C\colon \operatorname{lin}(\mathcal{F}) \to \mathbb{R}$ be any lower bounded, Lipschitz differentiable cost functional (that is, there exists $L > 0$ such that $\|\nabla C(F) - \nabla C(F')\| \leq L\|F - F'\|$ for all $F, F' \in \operatorname{lin}(\mathcal{F})$). Let $F_0, F_1, \ldots$ be the sequence of combined hypotheses generated

by the AnyBoost algorithm, using step-sizes

$$w_{t+1} := -\frac{\langle \nabla C(F_t), f_{t+1} \rangle}{L \| f_{t+1} \|^2}. \tag{12.9}$$

Then AnyBoost either halts on round t with $-\langle \nabla C(F_t), f_{t+1} \rangle \leq 0$, or $C(F_t)$ converges to some finite value C^*, in which case

$$\lim_{t \to \infty} \langle \nabla C(F_t), f_{t+1} \rangle = 0.$$

Proof First we need a general lemma.

Lemma 12.4
Let $(\mathcal{X}, \langle , \rangle)$ be an inner product space with squared norm $\|F\|^2 := \langle F, F \rangle$ and let $C: \mathcal{X} \to \mathbb{R}$ be a differentiable functional with $\|\nabla C(F) - \nabla C(F')\| \leq L \|F - F'\|$ for all $F, F' \in \mathcal{X}$. Then for any $w > 0$ and $F, G \in \mathcal{X}$,

$$C(F + wG) - C(F) \leq w \langle \nabla C(F), G \rangle + \frac{Lw^2}{2} \|G\|^2.$$

Proof Define $g: \mathbb{R} \to \mathbb{R}$ by $g(w) := C(F + wG)$. Then $g'(w) = \langle \nabla C(F + wG), G \rangle$ and hence

$$\begin{aligned} |g'(w) - g'(0)| &= |\langle \nabla C(F + wG) - \nabla C(F), G \rangle| \\ &\leq \|\nabla C(F + wG) - \nabla C(F)\| \|G\| \quad \text{by Cauchy-Schwartz} \\ &\leq Lw \|G\|^2 \quad \text{by Lipschitz continuity of } \nabla C. \end{aligned}$$

Thus, for $w > 0$,

$$g'(w) \leq g'(0) + Lw\|G\|^2 = \langle \nabla C(F), G \rangle + Lw\|G\|^2$$

which implies

$$\begin{aligned} g(w) - g(0) &= \int_0^w g'(\alpha)\, d\alpha \\ &\leq \int_0^w \langle \nabla C(F), G \rangle + L\alpha \|G\|^2 \, d\alpha \\ &= w \langle \nabla C(F), G \rangle + \frac{Lw^2}{2} \|G\|^2. \end{aligned}$$

Substituting $g(w) = C(F + wG)$ on the left hand side gives the result. ■

Now we can write:

$$\begin{aligned} C(F_t) - C(F_{t+1}) &= C(F_t) - C(F_t + w_{t+1} f_{t+1}) \\ &\geq -w_{t+1} \langle \nabla C(F_t), f_{t+1} \rangle - \frac{L w_{t+1}^2 \|f_{t+1}\|^2}{2} \quad \text{by Lemma 12.4.} \end{aligned}$$

If $\|f_{t+1}\| = 0$ then $\langle \nabla C(F_t), f_{t+1} \rangle = 0$ and AnyBoost will terminate. Otherwise,

the greatest reduction occurs when the right hand side is maximized, i.e., when

$$w_{t+1} = -\frac{\langle \nabla C(F_t), f_{t+1} \rangle}{L \|f_{t+1}\|^2},$$

which is the step-size in the statement of the theorem. Thus, for our stated step-size,

$$C(F_t) - C(F_{t+1}) \geq \frac{\langle \nabla C(F_t), f_{t+1} \rangle^2}{2L \|f_{t+1}\|^2}. \tag{12.10}$$

If $-\langle \nabla C(F_t), f_{t+1} \rangle \leq 0$ then AnyBoost terminates. Otherwise, since C is bounded below, $C(F_t) - C(F_{t+1}) \to 0$ which implies $\langle \nabla C(F_t), f_{t+1} \rangle \to 0$.

The next theorem shows that if the weak learner can always find the best weak hypothesis $f_t \in \mathcal{F}$ on each round of AnyBoost, and if the cost functional C is convex, then any accumulation point F of the sequence F_t generated by AnyBoost with step-sizes given by (12.9) is guaranteed to be a global minimum. It is convenient to assume that the hypothesis class $\mathcal{F}$ is *negation closed*, which means $f \in \mathcal{F}$ implies $-f \in \mathcal{F}$. In this case, a function f_{t+1} that maximizes $-\langle \nabla C(F_t), f_{t+1} \rangle$ always satisfies $-\langle \nabla C(F_t), f_{t+1} \rangle \geq 0$. For ease of exposition, we have assumed that rather than terminating when $-\langle \nabla C(F_T), f_{T+1} \rangle = 0$, AnyBoost simply continues to return F_T for all subsequent time steps t.

Theorem 12.5
Let $C\colon \text{lin}\,(\mathcal{F}) \to \mathbb{R}$ be a *convex* cost functional with the properties in Theorem 12.3, and let (F_t) be the sequence of combined hypotheses generated by the AnyBoost algorithm with step-sizes given by (12.9). Assume that the weak hypothesis class $\mathcal{F}$ is negation closed and that on each round the AnyBoost algorithm finds a function f_{t+1} maximizing $-\langle \nabla C(F_t), f_{t+1} \rangle$. Then the sequence (F_t) satisfies

$$\lim_{t \to \infty} \sup_{f \in \mathcal{F}} -\langle \nabla C(F_t), f \rangle = 0, \tag{12.11}$$

and any accumulation point F of F_t satisfies

$$C(F) = \inf_{G \in \text{lin}\,(\mathcal{F})} C(G). \tag{12.12}$$

Proof Equation (12.11) follows immediately from Theorem 12.3. For the proof of (12.12) we need the following more general lemma:

Lemma 12.6
Let C be a differentiable convex cost function on an inner product space $(\mathcal{X}, \langle , \rangle)$ with norm $\|F\|^2 = \langle F, F \rangle$. Let $\mathcal{M}$ be any linear subspace of $\mathcal{X}$ and let $\mathcal{M}^\perp$ denote the perpendicular subspace to $\mathcal{M}$ ($\mathcal{M}^\perp = \{G \in \mathcal{X}\colon \langle G, F \rangle = 0 \ \forall F \in \mathcal{M}\}$). If $F \in \mathcal{M}$ satisfies

$$\nabla C(F) \in \mathcal{M}^\perp$$

then

$$C(F) = \inf_{G \in \mathcal{M}} C(G).$$

Proof Consider $G \in \mathcal{M}$. By the convexity of C, for all $0 \leq \epsilon \leq 1$,

$$C\left((1-\epsilon)F + \epsilon G\right) - \left((1-\epsilon)C(F) + \epsilon C(G)\right) \leq 0.$$

Taking the limit as $\epsilon \to 0$ yields,

$\langle G - F, \nabla C(F) \rangle \leq C(G) - C(F)$.

Since $G - F \in \mathcal{M}$ and $\nabla C(F) \in \mathcal{M}^{\perp}$, this implies $C(G) \geq C(F)$. ■

Now let F be an accumulation point of F_t. By Lipschitz continuity of $\nabla C(F)$ and (12.11),

$$\sup_{f \in \mathcal{F}} - \langle \nabla C(F), f \rangle = 0,$$

which by the negation closure of $\mathcal{F}$ implies $\langle \nabla C(F), f \rangle = 0$ for all $f \in \mathcal{F}$, hence $\nabla C(F) \in \operatorname{lin}(\mathcal{F})^{\perp}$. Thus $F \in \operatorname{lin}(\mathcal{F})$ and $\nabla C(F) \in \operatorname{lin}(\mathcal{F})^{\perp}$, which by Lemma 12.6 implies (12.12).

12.5.2 Convergence of AnyBoost.L_1

The following theorem supplies a specific step-size for AnyBoost.L_1 and characterizes the limiting behaviour under this step-size regime.

Theorem 12.7
Let C be a cost function as in Theorem 12.3. Let $F_0, F_1, \ldots$ be the sequence of combined hypotheses generated by the AnyBoost.L_1 algorithm, using step-sizes

$$w_{t+1} := \frac{-\langle \nabla C(F_t), f_{t+1} - F_t \rangle}{L \| f_{t+1} - F_t \|^2 + \langle \nabla C(F_t), f_{t+1} - F_t \rangle} \tag{12.13}$$

Then AnyBoost.L_1 either terminates at some finite time t with $-\langle \nabla C(F_t), f_{t+1} - F_t \rangle \leq 0$, or $C(F_t)$ converges to a finite value C^*, in which case

$$\lim_{t \to \infty} \langle \nabla C(F_t), f_{t+1} - F_t \rangle = 0.$$

Proof Note that the step-sizes w_t are always positive. In addition, if the w_t are such that $\sum_{s=1}^{t} w_s < 1$ for all t then clearly the second case above will apply. So without loss of generality assume $\sum_{s=1}^{t} w_s = 1$. Applying Lemma 12.4, we have:

$$\begin{aligned} C(F_t) - C(F_{t+1}) &= C(F_t) - C\left(\frac{F_t + w_{t+1} f_{t+1}}{1 + w_{t+1}}\right) \\ &= C(F_t) - C\left(F_t + \frac{w_{t+1}}{1 + w_{t+1}} (f_{t+1} - F_t)\right) \\ &\geq -\frac{w_{t+1}}{1 + w_{t+1}} \langle \nabla C(F_t), f_{t+1} - F_t \rangle \end{aligned}$$

$$-\frac{L}{2}\left[\frac{w_{t+1}}{1+w_{t+1}}\right]^2 \|f_{t+1}-F_t\|^2. \tag{12.14}$$

If $-\langle \nabla C(F_t), f_{t+1}-F_t\rangle \leq 0$ then the algorithm terminates. Otherwise, the right hand side of (12.14) is maximized when

$$w_{t+1} = \frac{-\langle \nabla C(F_t), f_{t+1}-F_t\rangle}{L\|f_{t+1}-F_t\|^2 + \langle \nabla C(F_t), f_{t+1}-F_t\rangle}$$

which is the step-size in the statement of the theorem. Thus, for our stated step-size,

$$C(F_t) - C(F_{t+1}) \geq \frac{\langle \nabla C(F_t), f_{t+1}-F_t\rangle^2}{2L\|f_{t+1}-F_t\|^2},$$

which by the lower-boundedness of C implies $\langle \nabla C(F_t), f_{t+1}-F_t\rangle \to 0$. ■

The next theorem shows that if the weak learner can always find the best weak hypothesis $f_t \in \mathcal{F}$ on each round of AnyBoost.L_1, and if the cost function C is convex, then AnyBoost.L_1 is guaranteed to converge to the global minimum of the cost. As with Theorem 12.5, we have assumed that rather than terminating when $-\langle f_{T+1}-F_T, \nabla C(F_T)\rangle \leq 0$, AnyBoost.$L_1$ simply continues to return F_T for all subsequent time steps t.

Theorem 12.8

Let C be a *convex* cost function with the properties in Theorem 12.3, and let (F_t) be the sequence of combined hypotheses generated by the AnyBoost.L_1 algorithm using the step-sizes in (12.13). Assume that the weak hypothesis class $\mathcal{F}$ is negation closed and that on each round the AnyBoost.L_1 algorithm finds a function f_{t+1} maximizing $-\langle \nabla C(F_t), f_{t+1}-F_t\rangle$. Then

$$\lim_{t\to\infty} \sup_{f\in\mathcal{F}} -\langle \nabla C(F_t), f-F_t\rangle = 0, \tag{12.15}$$

and any accumulation point F of the sequence (F_t) satisfies

$$C(F) = \inf_{G\in \mathrm{co}(\mathcal{F})} C(G), \tag{12.16}$$

where $\mathrm{co}(\mathcal{F})$ is the set of all convex combinations of hypotheses from $\mathcal{F}$.

Proof Equation (12.15) follows immediately from Theorem 12.7. Now let F be an accumulation point of F_t. By (12.15) and continuity of $\nabla C(F)$, for all $f \in \mathcal{F}$,

$$\langle \nabla C(F), f-F\rangle = 0,$$

or equivalently $\langle \nabla C(F), f\rangle = \langle \nabla C(F), F\rangle$ for all $f \in \mathcal{F}$. Using the same argument as in the proof of Lemma 12.6, any $G \in \mathrm{co}(\mathcal{F})$ has

$$\langle G-F, \nabla C(F)\rangle \leq C(G) - C(F).$$

But because $\mathcal{F}$ is negation closed, we can write $G = \sum w_i f_i$ where all w_i are positive

and $\sum w_i = 1$. Then

$$\langle G - F, \nabla C(F) \rangle = \sum w_i \langle f_i, \nabla C(F) \rangle - \langle F, \nabla C(F) \rangle = 0.$$

It follows that $C(G) \geq C(F)$. ■

At this point we should note that for cost functions which are nonconvex (like those motivated by the theoretical result of Section 12.4) we can only guarantee convergence to a local minimum.

12.6 Experiments

AdaBoost had been perceived to be resistant to overfitting despite the fact that it can produce combinations involving very large numbers of classifiers. However, recent studies have shown that this is not the case, even for base classifiers as simple as decision stumps. Grove and Schuurmans [1998] demonstrated that running AdaBoost for hundreds of thousands of rounds can lead to significant overfitting, while a number of authors [Dietterich, 1998, Rätsch et al., 1998, Bauer and Kohavi, 1997, Maclin and Opitz, 1997] showed that, by adding label noise, overfitting can be induced in AdaBoost even with relatively few classifiers in the combination.

Given the theoretical motivations described in Sections 12.4 and 12.5 we propose a new algorithm (DOOM II) based on MarginBoost.L_1 which performs a gradient descent optimization of

$$\frac{1}{m} \sum_{i=1}^{m} (1 - \tanh(\lambda y_i F(\mathbf{x}_i))), \tag{12.17}$$

where F is restricted to be a convex combination of classifiers from some base class $\mathcal{F}$ and λ is an adjustable parameter of the cost function. Henceforth we will refer to (12.17) as the *normalized sigmoid cost function* (normalized because the weights are normalized so F is a convex combination). This family of cost functions (parameterized by λ) is qualitatively similar to the family of cost functions (parameterized by N) shown in Figure 12.1. Using the family from Figure 12.1 in practice may cause difficulties for the gradient descent procedure because the functions are very flat for negative margins and for margins close to 1. Using the normalized sigmoid cost function alleviates this problem.

Choosing a value of λ corresponds to choosing a value of the complexity parameter N in Theorem 12.2. It is a data dependent parameter which measures the resolution at which we examine the margins. A large value of λ corresponds to a high resolution and hence high effective complexity of the convex combination. Thus, choosing a large value of λ amounts to a belief that a high complexity classifier can be used without overfitting. Conversely, choosing a small value of λ corresponds to a belief that a high complexity classifier can only avoid overfitting if it has large margins.

Algorithm 12.6 : DOOM II

Require :

- A class of base classifiers $\mathcal{F}$ containing functions $f: X \to \{\pm 1\}$.
- A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$ with each $(\mathbf{x}_i, y_i) \in X \times \{\pm 1\}$.
- A weak learner $\mathcal{L}(S, D, F)$ that accepts a training set S, a distribution D on the training set and a combined classifier F, and returns base classifiers $f \in \mathcal{F}$ with small error: $\sum_{i=1}^{m} [F(\mathbf{x}_i) - f(\mathbf{x}_i)] y_i D(i)$.
- A fixed small step-size ϵ.

Let $D_0(i) := 1/m$ for $i = 1, \ldots, m$.
Let $F_0 := 0$.
for $t := 0$ to T **do**
  Let $f_{t+1} := \mathcal{L}(S, D_t, F_t)$.
  if $\sum_{i=1}^{m} D_t(i)[y_i f_{t+1}(\mathbf{x}_i) - y_i F_t(\mathbf{x}_i)] \leq 0$ **then**
    Return F_t.
  end if
  Let $w_{t+1} := \epsilon$.
  Let $F_{t+1} := \dfrac{F_t + w_{t+1} f_{t+1}}{\sum_{s=1}^{t+1} |w_s|}$.
  Let $D_{t+1}(i) := \dfrac{1 - \tanh^2(\lambda y_i F_{t+1}(\mathbf{x}_i))}{\sum_{i=1}^{m} (1 - \tanh^2(\lambda y_i F_{t+1}(\mathbf{x}_i)))}$
  for $i = 1, \ldots, m$.
end for

In the above implementation of DOOM II we are using a fixed small step-size ϵ (for all of the experiments $\epsilon = 0.05$). In practice the use of a fixed ϵ could be replaced by a line search for the optimal step-size at each round.

It is worth noting that since the l_1-norm of the classifier weights is fixed at 1 for each iteration and the cost function has the property that $C(-\alpha) = 1 - C(\alpha)$, the choice of λ is equivalent to choosing the l_1-norm of the weights while using the cost function $C(\alpha) = 1 - \tanh(\alpha)$.

Given that the normalized sigmoid cost function is nonconvex the DOOM II algorithm will suffer from problems with local minima. In fact, the following result shows that for cost functions satisfying $C(-\alpha) = 1 - C(\alpha)$, the MarginBoost.$L_1$ algorithm will strike a local minimum at the first step.

Lemma 12.9

Let $C: \mathbb{R} \to \mathbb{R}$ be any cost function satisfying $C(-\alpha) = 1 - C(\alpha)$. If MarginBoost.$L_1$ can find the optimal weak hypothesis f_1 at the first time step, it will terminate at the next time step, returning f_1.

Proof Assume without loss that $C'(0) < 0$. With $F_0 = 0$, $\langle \nabla C(F_0), f \rangle = (C'(0)/m) \sum_{i=1}^{m} y_i f(\mathbf{x}_i)$ and so by assumption, f_1 will satisfy

$$\sum_{i=1}^{m} y_i f_1(\mathbf{x}_i) = \inf_{f \in \mathcal{F}} \sum_{i=1}^{m} y_i f(\mathbf{x}_i)$$

and $F_1 = f_1$. Now $C(-\alpha) = 1 - C(\alpha) \Rightarrow C'(-\alpha) = C'(\alpha)$, and since f_1 only takes the values ± 1, we have for any f:

$$\langle \nabla C(F_1), f - F_1 \rangle = \frac{C'(1)}{m} \sum_{i=1}^{m} y_i (f(\mathbf{x}_i) - f_1(\mathbf{x}_i)).$$

Thus, for all $f \in \mathcal{F}$, $-\langle \nabla C(F_1), f - F_1 \rangle \leq 0$ and hence MarginBoost.L_1 will terminate, returning f_1. ■

A simple technique for avoiding this local minimum is to apply some notion of randomized initial conditions in the hope that the gradient descent procedure will then avoid this local minimum. Either the initial margins could be randomized or a random initial classifier could be chosen from $\mathcal{F}$. Initial experiments showed that both these techniques are somewhat successful, but could not guarantee avoidance of the single classifier local minimum unless many random initial conditions were tried (a computationally intensive prospect).

A more principled way of avoiding this local minimum is to remove f_1 from $\mathcal{F}$ after the first round and then continue the algorithm returning f_1 to $\mathcal{F}$ only when the cost goes below that of the first round. Since f_1 is a local minimum the cost is guaranteed to increase after the first round. However, if we continue to step in the best available direction (the flattest uphill direction) we should eventually "crest the hill" defined by the basin of attraction of the first classifier and then start to decrease the cost. Once the cost decreases below that of the first classifier we can safely return the first classifier to the class of available base classifiers. Of course, we have no guarantee that the cost will decrease below that of the first classifier at any round after the first. Practically however, this does not seem to be a problem except for very small values of λ where the cost function is almost linear over $[-1, 1]$ (in which case the first classifier corresponds to a global minimum anyway).

In order to compare the performance of DOOM II and AdaBoost a series of experiments were carried out on a selection of data sets taken from the UCI machine learning repository [Blake et al., 1998]. To simplify matters, only binary classification problems were considered. All of the experiments were repeated 100 times with 80%, 10% and 10% of the examples randomly selected for training, validation and test purposes respectively. The results were then averaged over the 100 repeats. For all of the experiments axis-orthogonal hyperplanes (also known as decision stumps) were produced by the weak learner. This fixed the complexity of the weak learner and thus avoided any problems with the complexity of the combined classifier being dependent on the actual classifiers produced by the weak learner.

For AdaBoost, the validation set was used to perform early stopping. AdaBoost was run for 2000 rounds and then the combined classifier from the round corresponding to minimum error on the validation set was chosen. For DOOM II, the validation set was used to set the data dependent complexity parameter λ. DOOM II was run for 2000 rounds with $\lambda = 2, 4, 6, 10, 15$ and 20 and the optimal λ was chosen to correspond to minimum error on the validation set after 2000 rounds. The typical behaviour of the test error as DOOM II proceeds is shown in Figure 12.2 for various values of λ. For small values of λ the test error converges to a value much worse than AdaBoost's test error. As λ is increased to the optimal value the test errors decrease. In the case of the `sonar` data set used in Figure 12.2 the test errors for AdaBoost and DOOM II with optimal λ are similar. Of course, with AdaBoost's adaptive step-size it converges much faster than DOOM II (which uses a fixed step-size).

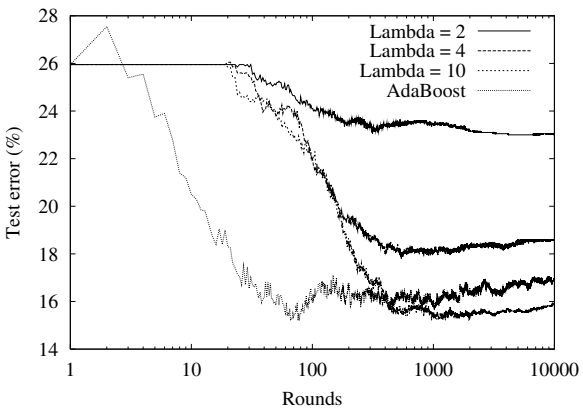


Figure 12.2 Test error for the `sonar` data set over 10000 rounds of AdaBoost and DOOM II with $\lambda = 2, 4$ and 10.

AdaBoost and DOOM II were run on nine data sets to which varying levels of label noise had been applied. A summary of the experimental results is provided in Table 12.2. The attained test errors are shown for each data set for a single stump, AdaBoost applied to stumps and DOOM II applied to stumps with 0%, 5% and 15% label noise. A graphical representation of the difference in test error between AdaBoost and DOOM II is shown in Figure 12.3. The improvement in test error exhibited by DOOM II over AdaBoost (with standard error bars) is shown for each data set and noise level. These results show that DOOM II generally outperforms AdaBoost and that the improvement is generally more pronounced in the presence of label noise.

The effect of using the normalized sigmoid cost function rather than the exponential cost function is best illustrated by comparing the cumulative margin distributions generated by AdaBoost and DOOM II. Figure 12.4 shows comparisons

		sonar	cleve	ionosphere	vote1	credit	breast-cancer	pima-indians	hypo1	splice
	Examples	208	303	351	435	690	699	768	2514	3190
	Attributes	60	13	34	16	15	9	8	29	60
0%	Stump	26.0	26.9	17.6	6.2	14.5	8.1	27.6	7.0	22.6
Label	AdaBoost	16.0	16.8	10.1	**3.5**	14.1	4.2	25.8	**0.5**	6.4
Noise	DOOM II	**15.8**	**16.5**	**9.7**	4.5	**13.0**	**3.0**	**25.1**	0.7	**5.7**
5%	Stump	30.4	29.0	21.7	10.6	18.0	12.1	29.7	12.4	26.4
Label	AdaBoost	**23.0**	21.6	16.7	9.6	17.5	9.0	**27.9**	8.6	13.9
Noise	DOOM II	23.3	**20.3**	**14.6**	**9.4**	**17.0**	**8.0**	**27.9**	**7.1**	**12.1**
15%	Stump	36.6	33.7	27.7	19.3	25.1	20.3	34.2	21.0	31.1
Label	AdaBoost	33.8	29.8	26.8	**19.0**	25.1	18.6	33.3	18.3	22.2
Noise	DOOM II	**32.6**	**27.6**	**25.9**	**19.0**	**24.7**	**17.6**	**33.1**	**17.1**	**20.3**

Table 12.2 Summary of test errors for a single stump, AdaBoost stumps and DOOM II stumps with varying levels of label noise on nine UCI data sets. The best test error for each data set is displayed in bold face. Note that since DOOM II uses an independent validation set to choose the cost function parameter λ, we are comparing it to a version of AdaBoost modified to use an independent validation set for early stopping.

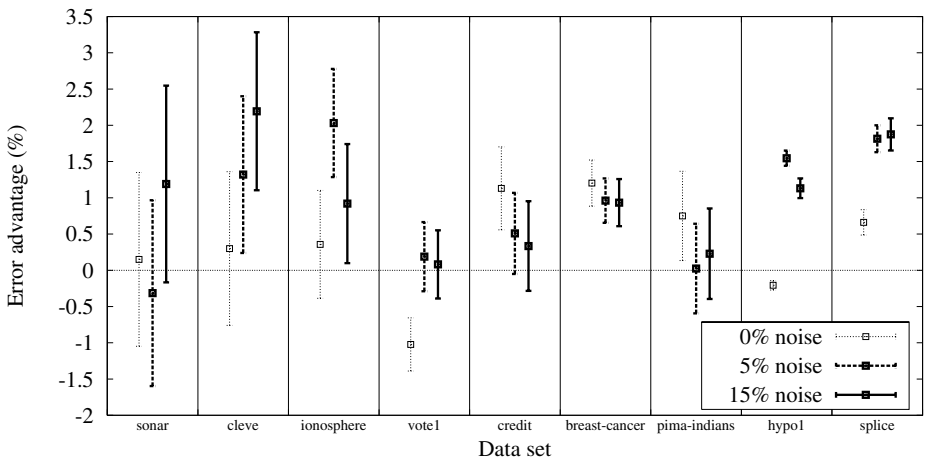


Figure 12.3 Summary of test error advantage (with standard error bars) of DOOM II over AdaBoost with varying levels of noise on nine UCI data sets.

for two data sets with 0% and 15% label noise applied. For a given margin, the value on the curve corresponds to the proportion of training examples with margin less than or equal to this value. These curves show that in trying to increase the

margins of negative examples AdaBoost is willing to sacrifice the margin of positive examples significantly. In contrast, DOOM II "gives up" on examples with large negative margin in order to reduce the value of the cost function.

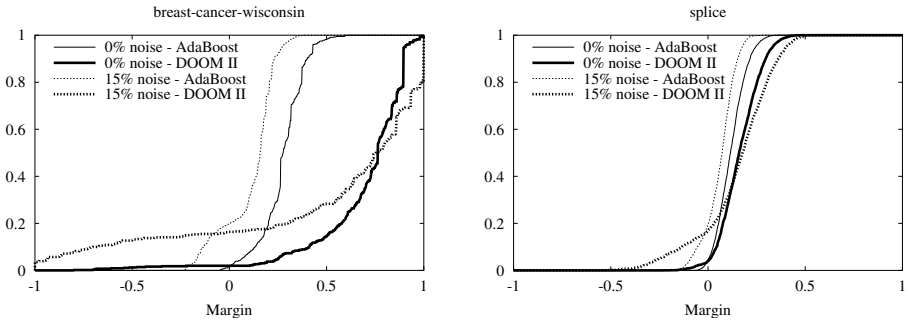


Figure 12.4 Margin distributions for AdaBoost and DOOM II with 0% and 15% label noise for the `breast-cancer` and `splice` data sets.

Given that AdaBoost suffers from overfitting and minimizes an exponential cost function of the margins, this cost function certainly does not relate to test error. How does the value of our proposed cost function correlate with AdaBoost's test error? The theoretical bound suggests that for the "right" value of the data dependent complexity parameter λ our cost function and the test error should be closely correlated. Figure 12.5 shows the variation in the normalized sigmoid cost function, the exponential cost function and the test error for AdaBoost for four UCI data sets over 10000 rounds. As before, the values of these curves were averaged over 100 random train/validation/test splits. The value of λ used in each case was chosen by running DOOM II for various values of λ and choosing the λ corresponding to minimum error on the validation set. These curves show that there is a strong correlation between the normalized sigmoid cost (for the right value of λ) and AdaBoost's test error. In all four data sets the minimum of AdaBoost's test error and the minimum of the normalized sigmoid cost very nearly coincide. In the `sonar` and `labor` data sets AdaBoost's test error converges and overfitting does not occur. For these data sets both the normalized sigmoid cost and the exponential cost converge, although in the case of the `sonar` data set the exponential cost converges significantly later than the test error. In the `cleve` and `vote1` data sets AdaBoost initially decreases the test error and then increases the test error (as overfitting occurs). For these data sets the normalized sigmoid cost mirrors this behaviour, while the exponential cost converges to 0.

To examine the effect of step-size we compare AdaBoost to a modified version using fixed step-sizes, called ε-AdaBoost. In ε-AdaBoost, the first classifier is given weight 1 and all others thereafter are given weight ε. A comparison of the test errors of both of these algorithms for various values of ε is shown in Figure 12.6.

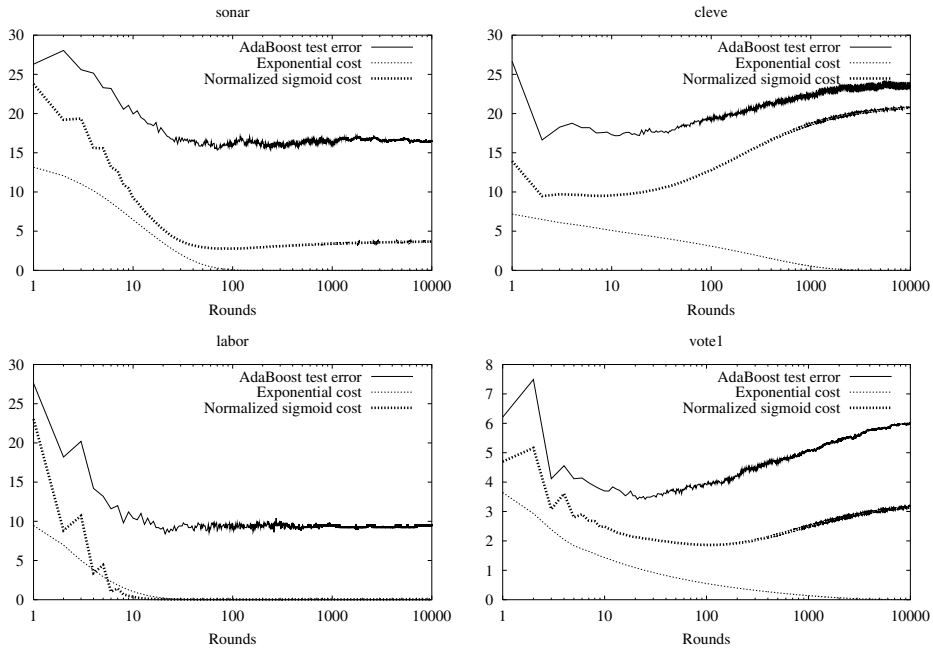


Figure 12.5 AdaBoost test error, exponential cost and normalized sigmoid cost over 10000 rounds of AdaBoost for the `sonar`, `cleve`, `labor` and `vote1` data sets. Both costs have been scaled in each case for easier comparison with test error.

As expected, changing the value of the fixed step size ε simply translates the test error curve on the log scale and does not significantly alter the minimum test error.

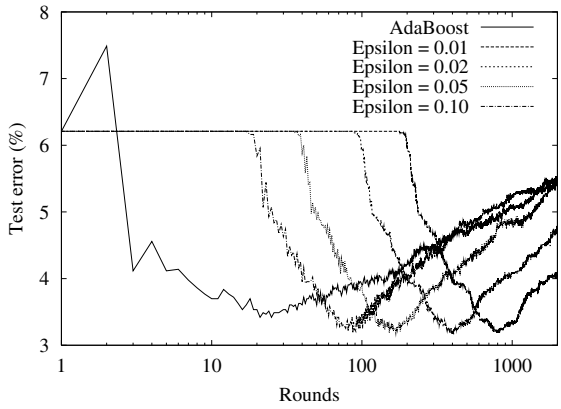


Figure 12.6 Test error for the `vote1` data set over 2000 rounds of AdaBoost and ε-AdaBoost for $\varepsilon = 0.01, 0.02, 0.05$ and 0.10.

12.7 Conclusions

We have shown how most existing "boosting-type" algorithms for combining classifiers can be viewed as gradient descent on an appropriate cost functional in a suitable inner product space. We presented AnyBoost, an abstract algorithm of this type for generating general linear combinations from some base hypothesis class, and a related algorithm, AnyBoost.L_1, for generating convex combinations from the base hypothesis class. Prescriptions for the step-sizes in these algorithms guaranteeing convergence to the optimal linear or convex combination were given.

For cost functions depending only upon the *margins* of the classifier on the training set, AnyBoost and AnyBoost.L_1 become MarginBoost and MarginBoost.L_1. We showed that many existing algorithms for combining classifiers can be viewed as special cases of MarginBoost.L_1; each algorithm differing only in its choice of margin cost function and step-size. In particular, AdaBoost is MarginBoost.L_1 with e^{-z} as the cost function of the margin z, and with a step-size equal to the one that would be found by a line search.

The main theoretical result from [Mason et al., 1999] provides bounds on the generalization performance of a convex combination of classifiers in terms of training sample averages of certain, sigmoid-like, cost functions of the margin. This suggests that algorithms such as Adaboost that optimize an exponential margin cost function are placing too much emphasis on examples with large negative margins, and that this is a likely explanation for overfitting, particularly in the presence of label noise.

Motivated by this result, we derived DOOM II—a further specialization of MarginBoost.L_1—that used $1-\tanh(z)$ as its cost function of the margin z. Experimental results on the UCI datasets verified that DOOM II generally outperformed AdaBoost when boosting decision stumps, particularly in the presence of label noise. We also found that DOOM II's cost on the training data was a very reliable predictor of test error, while AdaBoost's exponential cost was not.

In future we plan to investigate the properties of AnyBoost.L_2, mentioned in Sec. 12.2.3. Although we do not have theoretical results on the generalization performance of this algorithm, viewed in the inner product space setting an L_2 constraint on the combined hypothesis is considerably more natural than an L_1 constraint. In addition, the inner product perspective on boosting can be applied to any inner product space, not just spaces of functions as done here. This opens up the possibility of applying boosting in many other machine learning settings.

Acknowledgments

This research was supported by the ARC. Llew Mason was supported by an Australian Postgraduate Research Award. Jonathan Baxter was supported by an Australian Postdoctoral Fellowship. Peter Bartlett and Marcus Frean were supported by an Institute of Advanced Studies/Australian Universities Collaborative grant. Thanks to Shai Ben-David for a stimulating discussion.

13 Towards a Strategy for Boosting Regressors

Grigoris Karakoulas
Global Analytics Group
Canadian Imperial Bank of Commerce
161 Bay St., BCE-11,
Toronto ON
Canada M5J 2S8
karakoul@cibc.ca
http://www.cs.toronto.edu/∼grigoris

John Shawe-Taylor
Department of Computer Science
Royal Holloway, University of London
Egham, Surrey TW20 0EX
UK
j.shawe-taylor@dcs.rhbnc.ac.uk
http://www.cs.rhbnc.ac.uk/people/staff/shawe-taylor.shtml

We study the problem of boosting learners for regression using the perspective afforded by a margin analysis of classifier boosting. The approach motivates a novel strategy for generating the distribution of examples used to train the weak learners, and for determining their target values. Experimental results are given to show the performance of our approach.

13.1 Introduction

The idea of boosting learners who can perform slightly better than random in a classification task led to a general purpose algorithm which is able to combine the strengths of different hypotheses [Schapire, 1990]. This approach to combining the strengths of individual learners was further refined in the Adaboost algorithm which created a weighted combination, with the weights determined by the performance of the individual weak learners [Freund and Schapire, 1997]. This approach to improving the performance of classification algorithms has been incorporated into state-of-the-art learning systems and has received attention from the statistical community [Friedman et al., 1998].

The problem of applying a similar strategy for regression has also begun to be studied, initially by reducing to the classification case [Bertoni et al., 1997, Freund and Schapire, 1997]. The disadvantages of this approach include the implementation overhead, the massive weight changes when errors are small compared with the bounds of the target and that it may be inappropriate for algorithms which rely on the gradient of the error function. Ridgeway et al. [1998] also transformed the regression problem to a classification one and fitted a naive Bayes classifier to the resulting dataset. The initial results from this technique do not seem to counterbalance its computational complexity. Drucker [1997] developed a boosting algorithm by devising a loss for each training point as the ratio of the error over the maximal error and using multiplicative weight updating. The main disadvantage of this algorithm is the dependence of the loss function on the maximal error. This may lead to big changes in the weighting as a single extreme value varies.

The results of these approaches were not very encouraging and it appeared that the very impressive improvements found in the classification case were not reproducible for regression. In a very recent paper Friedman [1999] has developed a set of boosting algorithms for regression from the perspective of numerical optimization in function space. His algorithms deviate from standard boosting by performing deterministic gradient descent in the space of weak learners, rather than stochastic gradient descent in boosting where the distribution of examples is also used to control the generation of the weak learners.

We study the problem of boosting learners for regression and motivate a novel strategy for generating the distribution of examples used to train the weak learners, and for determining their target values. Our approach is motivated by three ideas.

First, we build on recent results bounding the generalization in terms of the margin of a classifier. This viewpoint has been used to explain the performance of the standard boosting algorithm [Schapire et al., 1998], where it is shown that the distribution given to the weak learners is a function of the margins of the training points with respect to the current hypothesis. More recently the analysis of generalization in terms of the margin has been extended to more robust measures of the distribution of margin values [Shawe-Taylor and Cristianini, 1998] with applications to regression. Using this perspective on the boosting procedure, we

motivate a different criterion and distribution for the weak learners which are then combined to optimize the overall performance for future accuracy.

Second, as in [Schapire and Singer, 1998], we cast the boosting regression problem as an optimization one. Thirdly, to control the complexity of the weak learners, we expand on the constructive neural network algorithm for incremental learning of Dunkin et al. [1997].

The rest of the chapter is structured as follows. Section 13.2 presents some background theoretical results. The boosting strategy is developed in Section 13.3. The algorithm for generating weak learners is presented in Section 13.4, while the overall boosting algorithm is outlined in Section 13.5. Experimental results are given in Section 13.6 that illustrate the performance of our approach. The chapter is concluded in Section 13.7.

13.2 Background Results

Our main ingredient in motivating the new algorithm will be Theorem 13.3 which bounds the probability that a regression function makes an error larger than θ on a randomly generated test example. The aim of the boosting algorithm will be to minimize this error bound. In order to give the theorem, recall the notion of γ-shattering from the introduction (Definition 1.7).

The first bound on the fat shattering dimension of bounded linear functions in a finite dimensional space was obtained by Shawe-Taylor et al. [1998]. Gurvits [1997] generalized this to infinite dimensional Banach spaces. We will quote an improved version of this bound for Hilbert spaces which is contained in [Bartlett and Shawe-Taylor, 1999] (see also Theorem 1.12, slightly adapted here for an arbitrary bound on the linear operators).

Theorem 13.1 Bartlett and Shawe-Taylor [1999]
Consider a Hilbert space and the class of linear functions L of norm less than or equal to A restricted to the sphere of radius R about the origin. Then the fat shattering dimension of L can be bounded by

$$\mathrm{Fat}_L(\gamma) \leq \left(\frac{AR}{\gamma}\right)^2.$$

Definition 13.2
We say that a class of functions $\mathcal{F}$ is *sturdy* if and only if its images under the evaluation maps

$$\tilde{x}_{\mathcal{F}}\colon \mathcal{F} \longrightarrow \mathbb{R}, \quad \tilde{x}_{\mathcal{F}}\colon f \mapsto f(x) \tag{13.1}$$

are compact subsets of $\mathbb{R}$ for all $x \in X$.

Fix $\theta \geq 0$. For a training point $(x, y) \in X \times \mathbb{R}$, real valued function f and $\gamma \in \mathbb{R}$, we define

$$\partial((x, y), f, \gamma) = \max\{0, |f(x) - y| - (\theta - \gamma)\}. \tag{13.2}$$

This quantity is the amount by which f exceeds the error margin $\theta - \gamma$ on the point (x, y) or 0 if f is within $\theta - \gamma$ of the target value. Hence, this is the ϵ-insensitive loss measure considered by Vapnik [1995] with $\epsilon = L^* := \theta - \gamma$. For a training set S, we define (cf. (19.1))

$$\mathcal{D}(S, f, \gamma) = \sqrt{\sum_{(x,y)\in S} \partial((x, y), f, \gamma)^2}. \tag{13.3}$$

We now quote a result from [Shawe-Taylor and Cristianini, 1998] (see also Chapter 19). For more background material on the motivation of Support Vector Regression using these bounds we refer the reader to [Cristianini and Shawe-Taylor, 2000].

Theorem 13.3 Shawe-Taylor and Cristianini [1998]
Let $\mathcal{F}$ be a sturdy class of real-valued functions with range $[a, b]$ and fat shattering dimension bounded by $\mathrm{Fat}_{\mathcal{F}}(\gamma)$. Fix $\theta \in \mathbb{R}$, $\theta > 0$ and a scaling of the output range $\eta \in \mathbb{R}$. Consider a fixed but unknown probability distribution on the space $X \times [a, b]$. Then with probability $1 - \delta$ over randomly drawn training sets S of size m for all γ with $\theta \geq \gamma > 0$ for any function $f \in \mathcal{F}$ the probability that f has output error $|f(x) - y|$ larger than θ on a randomly chosen input/output pair (x, y) is bounded by

$$\epsilon(m, k, \delta) = \frac{2}{m}\left(k \log_2 65m \left(\frac{b-a}{\gamma}\right)^2 \log_2 9em \left(\frac{b-a}{\gamma}\right) + \log_2 \frac{64m^{1.5}(b-a)}{\delta\eta}\right),$$

where

$$k = \left[\mathrm{Fat}_{\mathcal{F}}(\gamma^-/16) + \tilde{\mathcal{D}}^2\right] \quad \text{and} \quad \tilde{\mathcal{D}} = 16(\mathcal{D}(S, f, \gamma) + \eta)/\gamma,$$

provided $m \geq 2/\epsilon$ and there is no discrete probability on training points with error greater than θ.

There has been some work on introducing soft margins into boosting in order to avoid overfitting. For example Rätsch et al. [1998] used a postprocessing phase to choose a soft margin hyperplane in place of the weights found by boosting. Our strategy is to start with the soft margin bound on generalization of Theorem 13.3 and motivate a boosting algorithm which boosts the soft margin directly rather than the more usual exponential function of the margin. This should make the algorithm more resilient to overfitting.

Combining Theorem 13.1 and 13.3 suggests that the quantity which will bound the probability of a randomly generated point having error greater than θ in the case of linear function classes with weight vector norm B and inputs in a ball of

radius R about the origin is

$$\frac{B^2R^2 + \mathcal{D}(S, f, \gamma)^2}{\gamma^2}. \tag{13.4}$$

In the next section we will use this equation to optimize the choice of γ, hence determining the ϵ-insensitive measure as $L^* = \theta - \gamma$. The aim is to optimize the bound by trading off the complexity against the training error. As the boosting progresses this trade-off is adjusted automatically to optimize the bound. Although we do not do so, there would be no difficulty in using Theorem 13.3 to give a bound on the overall performance of the boosted regressor.

13.3 Top Level Description of the Boosting Strategy

The top level algorithm looks like this:

Define $(x)_+ = x$ if $x \geq 0$, 0 otherwise. Following the motivation suggested in the previous section, we then want to minimize:

$$\mathcal{D}(S, f, \theta - L^*)^2 = \sum_i (|f(x_i) - y_i| - L^*)_+^2 \tag{13.5}$$

for an appropriate choice of L^*, which will be chosen adaptively $L^* = L_t^*$ at the t-th boosting step. We define

$$Z_t = \frac{\sum_i (|f_t(x_i) - y_i| - L_t^*)_+^2}{\sum_i (|f_{t-1}(x_i) - y_i| - L_{t-1}^*)_+^2} \tag{13.6}$$

where $f_t(x) = f_{t-1}(x) + \alpha_t h_t(x)$ and

$$Z_1 = \frac{\sum_i (|f_1(x_i) - y_i| - L_1^*)_+^2}{m}. \tag{13.7}$$

The standard approach taken by boosting is to greedily minimize Z_t at stage t. Friedman et al. [1998] motivate the standard choice of distribution by showing that it implements a gradient descent in the space of weak learners. For boosting regressors there are two ways in which the remaining errors appear, in the weight given to the training examples in the distribution used for the next weak learner, and in the residual errors which are also passed to the weak learner as target values. Following [Friedman et al., 1998] we take the derivative of Z_t with respect to the loss of the current composite hypothesis on the i-th training example to define the distribution D^{t+1} used for training the $t+1$-st weak learner. Hence, we define

$$D_i^{t+1} \propto (|f_t(x_i) - y_i| - L_t^*)_+, \tag{13.8}$$

and take $D_i^0 = 1/m$. Note that this means that the weak learner may have zero weight on some training examples. Using the definition of Z_t given above we have

$$\mathcal{D}(S, f, \theta - L^*)^2 = \sum_i (|f_t(x_i) - y_i| - L_t^*)_+^2 = m \prod_{j=1}^{t} Z_j. \tag{13.9}$$

Note that as the iterations proceed the value of L_t^* reduces, so that we approach iterative least squares regression, albeit with an alternative weighting of the examples. The value of L_t^* is that it helps to eliminate from consideration points whose error is already well controlled hence focussing the attention of the weak learner on the examples that are proving difficult to fit.

At each stage our strategy is to choose α_t to minimize Z_t. This is equivalent to minimizing the expression:

$$f_t(\alpha) = \sum_i (|f_{t-1}(x_i) + \alpha h_t(x_i) - y_i| - L_t^*)_+^2, \tag{13.10}$$

with respect to α, and setting $\alpha_t = \text{argmin}\,(f_t(\alpha))$. We define

$$s_i(\alpha) = \begin{cases} -1 & \text{if } f_{t-1}(x_i) + \alpha h_t(x_i) - y_i < -L_t^* \\ 1 & \text{if } f_{t-1}(x_i) + \alpha h_t(x_i) - y_i > L_t^* \\ 0 & \text{otherwise.} \end{cases} \tag{13.11}$$

Hence, we can write

$$f(\alpha) = \sum_i s_i(\alpha)(f_{t-1}(x_i) + \alpha_t h_t(x_i) - y_i - s_i(\alpha) L_t^*)^2. \tag{13.12}$$

In order to choose α_t to minimize this sum (following the approach of Schapire and Singer [1998]), we will have to divide the interval into 2m segments since there will be critical points when $s_i(\alpha)$ changes value. Hence critical values of α are $\alpha_i = (L_t^* + y_i - f_{t-1}(x_i))/h_t(x_i)$ and $\alpha_i^* = (y_i - f_{t-1}(x_i) - L_t^*)/h_t(x_i)$. Once inside an interval the optimization will be quadratic so that an exact solution can be obtained. The algorithm examines all the critical points and optimizes in the two adjacent intervals.

Note that for $L^* < \gamma$, the results of the previous section show that the following expression:

$$\begin{aligned} &\frac{A^2R^2}{(\theta - L^*)^2} + \sum_{i:L^*<|f(x_i)-y_i|} (L^* - |f(x_i) - y_i|)^2/(\theta - L^*)^2 \\ &= \frac{A^2R^2}{\gamma^2} + \sum_{i:L^*<|f(x_i)-y_i|} (1 + (|f(x_i) - y_i| - \theta)/\gamma)^2 \end{aligned} \tag{13.13}$$

where $\gamma = \theta - L^*$, can be used to bound the probability of a randomly drawn test point having error greater than θ. One can think of this bound as a way of trading in the excess errors (over L^*) for additional complexity. The quantity A^2R^2/γ^2 is the base complexity, where L^* determines the band within which we consider the accuracy sufficient. The larger γ becomes the more accuracy we demand, which reduces the base complexity but increases the cost of the excess errors.

The training set is used to choose $\theta = \theta_t$ – this is the target error that we are trying to reduce. The value taken is the 95 percentile of the errors on the training set. Once determined we choose L_t^* to be the value which minimizes the above expression subject to the constraint that $L_t^* < \theta$.

To summarize the top level algorithm: Consider an arbitrary stage t. At this stage we will have a current hypothesis, f_t, given by $f_t = \sum_{j=1}^{t} \alpha_j h_j$. This hypothesis will have residual errors $r_i = y_i - f_t(x_i)$; and we let θ_t be the 95 percentile of the set $\{r_i\}$. We then define $A = \sqrt{\sum_{j=1}^{t} \alpha_j^2}$; $R_i = \sqrt{\sum_{j=1}^{t} h_j(x_i)^2}$; and set $R = \max_i R_i$. Now we choose L_{t+1}^* to be the value of L which minimizes the expression

$$\frac{A^2 R^2}{(\theta_t - L)^2} + \frac{\sum_{i:L<|f_t(x_i)-y_i|}(L - |f_t(x_i) - y_i|)^2}{(\theta_t - L)^2}, \tag{13.14}$$

and set the distribution D^{t+1} according to

$$D_i^{t+1} \propto (|f_t(x_i) - y_i| - L_{t+1}^*)_+. \tag{13.15}$$

The $(t+1)$-st weak learner h_{t+1} will then be generated using a sample generated according to D^{t+1} using target values given by the current residuals r_i. Finally, α_{t+1} is chosen to be the value of α which minimizes the expression

$$\sum_i (|f_t(x_i) + \alpha h_{t+1}(x_i) - y_i| - L_{t+1}^*)_+^2. \tag{13.16}$$

Algorithm 13.1 : Weak real learner

Require :

- A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$.
- A bound, "factor".

Let $B = \max(|y_i|) * \text{factor}$.
TotB $= 0$.
$j = 1$.
while error$(j-1)$ − error$(j) > 10^{-4}$ and $j \leq$ maxHiddenNodes **do**
  TotB $=$ TotB $* (j-1)/j$.
  for $k = 1, \ldots, j-1$ **do**
    $b_k = b_k * (j-1)/j$.
  end for
  for $i = 1, \ldots, m$ **do**
    Initialize $fn_j(\mathbf{x}_i)$ to zero.
    for $k = 1, \ldots, j-1$ **do**
      $fn_j(\mathbf{x}_i) = fn_j(\mathbf{x}_i) + b_k g_k(\mathbf{x}_i)$; % where g_k is the kth sigmoid output.
    end for
    Calculate residuals using the sum of the network outputs; $r_i = y_i - fn_j(\mathbf{x}_i)$.
  end for
  Bmax $= B -$ TotB.
  Train new network $b_j * g_j$ to output r_i adjusting neuron weights and output weight b_j which is constrained by adding $(|b_j| - Bmax)_+^2$ to error.
  TotB $=$ TotB $+ |b_j|$.
  error$(j) = 1/N * \sum_i (fn_j(\mathbf{x}_i) + b_j g_j(\mathbf{x}_i) - y_i)^2$.
  $j = j + 1$.
end while
Return $h = \sum_{k=1}^{j-1} b_k g_k$.

13.4 Generation of Weak Learners

We wished to allow the weak learners to alter their complexity by for example varying the number of hidden units in a neural network. In order to allow us to control complexity in a continuous fashion (rather than a discrete number of hidden units) we moved to using the incremental learning algorithm described by Dunkin et al. [1997]. This algorithm outputs a single hidden layer network of sigmoid neurons with a linear output unit. The complexity is controlled by a bound B placed on the sum of the weights connecting to the output unit. The algorithm is guaranteed to converge to a global optimum of the L_2 error of its output provided the neuron added at each incremental step is chosen optimally. Since a boosting algorithm creates a linear combination of its weak learners, the overall combined learner will be equivalent to a single layer neural network with sigmoid units. We could attempt to find such a neural network by applying the weak learning algorithm once with a very large value of B. In our experiments we compare results obtained following this approach with using the boosting strategy. It should, however, be clear that the boosting is providing an alternative way of searching the hypothesis space by focussing the algorithms attention on the examples that are consistently proving hard to fit.

The complexity of the weak learner is controlled by the bound B placed on the sum of the weights connecting to the output neuron of the weak learner. We are using $B = \text{factor} \times \max|y|$, with factor controlled by the progress of the approximation. If the approximation is improving by more than 5% as each weak learner is added the factor is reduced slightly, while the reverse holds if the approximation improves by less than 5%. The overall incremental algorithm is given in Algorithm 13.1.

13.5 Overall Algorithm

We are now able to present the overall boosting algorithm as shown in Algorithm 13.2. Here the final regressor will be a three layer sigmoid neural network with linear output neuron and a tree like structure for the computational nodes. This structure is more powerful than that output by the support vector machine with a sigmoid kernel (assuming that the sigmoid kernel is positive definite for the given data points and hence could be used).

The next section will present preliminary results obtained with the above algorithm on the Boston housing dataset, including comparisons with other regression stategies, including ν-support vector regression and bagging.

Algorithm 13.2 : RealBoost

Require :

- A training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$.

Initialize L_1^*, θ_1, factor, $f_0 = 0$.
Initialize D^0 to be the uniform distribution over the $\mathbf{x}'s$.
$t = 1$, $A = 0$.
Let $r_i = y_i$, for $i = 1, \ldots, m$.
while $t \leq 50$ **do**
  Select a sample $(\mathbf{x}_{k_1}, r_{k_1}), \ldots, (\mathbf{x}_{k_\ell}, r_{k_\ell})$ according to distribution D^{t-1}.
  $h_t = $ Weak real learn$((\mathbf{x}_{k_1}, r_{k_1}), \ldots, (\mathbf{x}_{k_\ell}, r_{k_\ell}),$ factor$)$.
  $\alpha_t = \text{argmin}_\alpha \left\{\sum_i (|f_{t-1}(\mathbf{x}_i) + \alpha h_t(\mathbf{x}_i) - y_i| - L_t^*)_+^2\right\}$.
  $f_t = f_{t-1} + \alpha_t h_t$.
  $A = \sqrt{\alpha_t^2 + A^2}$.
  Let $r_i = r_i - \alpha_t h_t(\mathbf{x}_i)$, for $i = 1, \ldots, m$.
  if max train error reduction $> 5\%$ **then**
    Decrement factor.
  else
    Increment factor.
  end if
  $\theta_t = 95$ percentile of errors of f_t on the training set.
  $R = \max_i(\|(h_j(\mathbf{x}_i))_{j=1}^t\|)$.
  $L_{t+1}^* = \text{argmin}_{L<\theta_t} \left\{\frac{A^2R^2}{(\theta_t - L)^2} + \frac{\sum_{i:L<|f_t(\mathbf{x}_i)-y_i|}(L-|f_t(\mathbf{x}_i)-y_i|)^2}{(\theta_t - L)^2}\right\}$.
  Compute a distribution $D^{t+1}(x_i) \propto (|f_t(\mathbf{x}_i) - y_i| - L_{t+1}^*)_+$.
  $t = t + 1$.
end while
Return f_{t-1}.

13.6 Experiments

The experiments presented here are preliminary and are intended to show that the approach compares well with bagging [Breiman, 1996] and Drucker's algorithm [Drucker, 1997], and delivers impressive results on one benchmark dataset.

The data considered was the Boston housing dataset [Merz and Murphy, 1998], which comprises 506 cases, the dependent variable being the median price of housing in the Boston area. There are 12 continuous predictor variables. The data was split into 481 cases for training, and 25 for testing. In all experiments reported the algorithms were run over 100 random splits of training and testing sets.

Figure 13.1 shows the average decay in training and testing error of RealBoost with boosting iteration. Note that the soft margin approach does seem to have the desired effect of controlling overfitting, since the generalization error decays consistently downwards, even after the complexity of the weak learners starts to increase (see Figure 13.4). Figure 13.2 shows how the value of L_t^* decays with boosting iteration, while the complexity of the boosted regressor ($\sum_t \alpha_t^2$) increases.

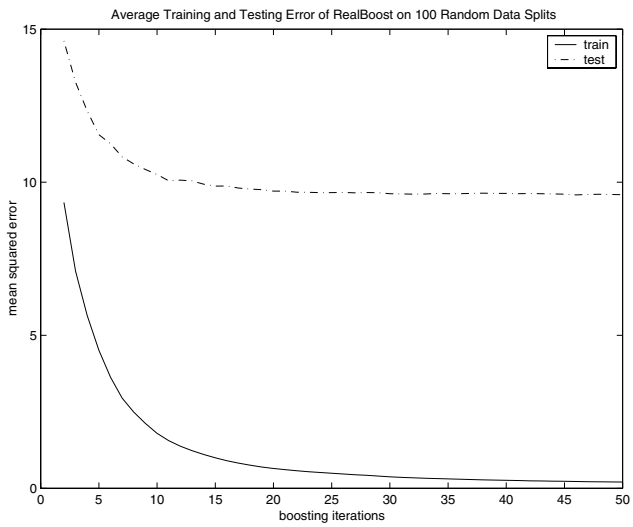


Figure 13.1 Test and training error against boosting iteration of RealBoost

Figure 13.3 shows the movement of the values of L_t^* and θ_t with the progress of the boosting iterations. Finally Figure 13.4 shows the development of the value of "factor," which controls the capacity of the weak learners.

To compare our algorithm with bagging, we trained 50 weak learners, according to the incremental algorithm of Section 13.4, by resampling uniformly from the training set. We implemented Drucker's algorithm by similarly training 50 weak learners. As discussed in Section 13.4 we also need to test whether the boosted regressor is a "better" solution than the one provided by fitting a single, large neural network. For this purpose, we also ran experiments by using the incremental algorithm to train a single weak learner, where the capacity of the learner was set to an arbitrarily high number.

The results are summarized in Table 13.1. Compared to the other algorithms RealBoost has smaller standard deviation by a factor of 2. It is worth noting that bagging and the single weak learner yielded very similar results. This is in contrast to the results from RealBoost, indicating that despite its somewhat ad hoc design the reweighting scheme of RealBoost may give rise to a combined regressor with lower generalization error.

The results from our algorithm compare well with the results from bagging regression trees in [Drucker, 1997] where the average test error over 100 random splits is 12.4 and the results from ν-SVR in [Schölkopf et al., 1998c] where for different values of the parameter ν the average test of support vector regression varies between 8.7 (std = 6.8) and 11.3 (std = 9.5). It should be pointed out that in the latter algorithm the parameter ν that controls the accuracy and number of support vectors has to be specified a priori. Despite this the experiments are performed using both training and validations sets for training. We also used the validation set for training, but did not need to adapt any parameters, since θ and

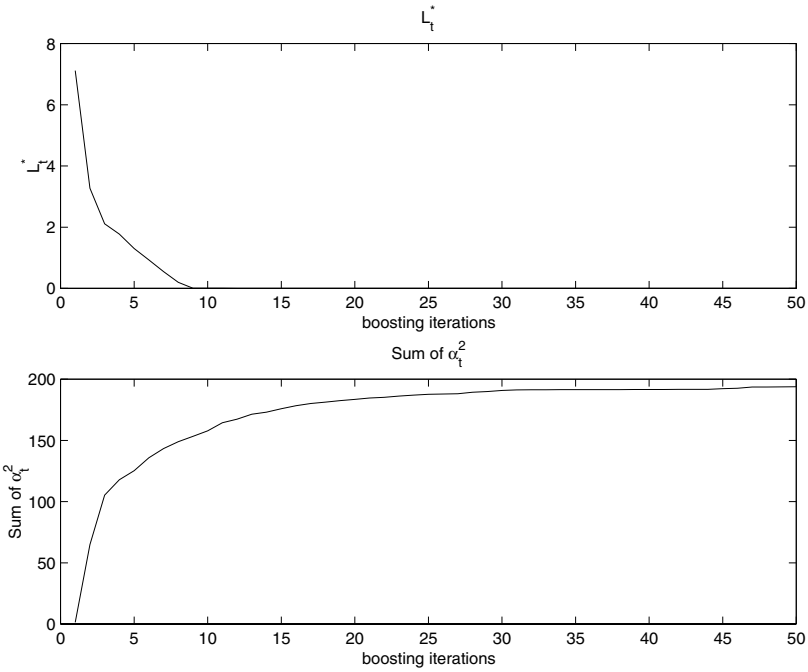


Figure 13.2 Decay in L_t^* and increase in Sum of α_t^2

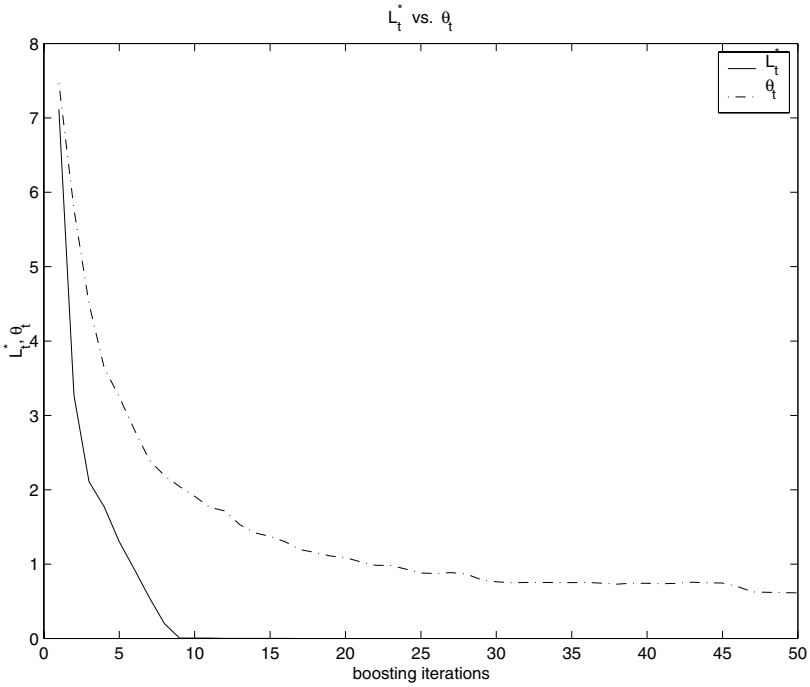


Figure 13.3 Progress of θ_t and L_t^*

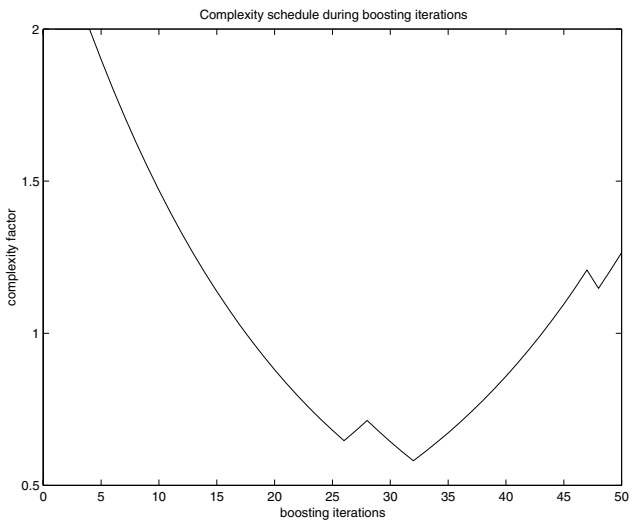


Figure 13.4 Complexity schedule during the boosting iterations

	Training Mean (STD)	Testing Mean (STD)
RealBoost	0.204 (0.992)	9.592 (5.256)
Drucker	13.458 (1.354)	16.53 (10.82)
Bagging	11.712 (0.403)	14.704 (10.275)
Single learner	11.885 (1.771)	15.576 (10.51)

Table 13.1 Mean and standard deviations of RealBoost, Drucker's algorithm, Bagging and the single weak learner

L^* are adjusted automatically by the algorithm. The setting of the value of θ_t as the 95% percentile and the reduction factor could require adaptation for significantly different types of data, but the one setting was adequate for all of the experiments reported here and hence we regard these as prefixed constant values.

13.7 Conclusions

The chapter has presented a strategy for boosting weak regressors that is motivated by recent results estimating the probability of errors based on a measure of the distribution of the training errors. The bound provides a guide to setting the insensitivity of the loss function used, which is then used to generate the distribution of examples passed to the weak learner.

Experiments have demonstrated that the algorithm performs well on a standard benchmark dataset, when compared with alternative approaches.

IV Leave-One-Out Methods

14 Bounds on Error Expectation for SVM

Vladimir Vapnik
AT&T Labs
Red Bank, NJ 07701
vlad@research.att.com

Olivier Chapelle
École Normale Supérieure de Lyon
69364 Lyon Cedex 07, France
ochapell@ens-lyon.fr

We introduce the concept of span of support vectors (SV) and show that the generalization ability of support vector machines (SVM) depends on this new geometrical concept. We prove that the value of the span is always smaller (and can be much smaller) than the diameter of the smallest sphere containing the support vectors, used in previous bounds [Vapnik, 1998]. We also demonstrate experimentally that the prediction of the test error given by the span is very accurate and has direct application in model selection (choice of the optimal parameters of the SVM).

14.1 Introduction

Recently, a new type of algorithm with a high level of performance called Support Vector Machines (SVM) has been introduced [Boser et al., 1992, Vapnik, 1995].

Usually, the good generalization ability of SVM is explained by the existence of a large margin: bounds on the error rate for a hyperplane that separates the data with some margin were obtained in [Bartlett and Shawe-Taylor, 1999, Shawe-Taylor et al., 1998]. In Vapnik [1998], another type of bound was obtained which demonstrated that for the separable case the expectation of probability of error for hyperplanes passing through the origin depends on the expectation of R^2/ρ^2, where R is the maximal norm of support vectors and ρ is the margin.

In this chapter we derive bounds on the expectation of error for SVM from the *leave-one-out* estimator, which is an unbiased estimate of the probability of test error. These bounds (which are tighter than the one defined in Vapnik [1998] and valid for hyperplanes not necessarily passing through the origin) depend on a new concept called *the span of support vectors*.

The bounds obtained show that the generalization ability of SVM depends on more complex geometrical constructions than large margin. To introduce the concept of the span of support vectors we have to describe the basics of SVM.

14.2 SVM for Pattern Recognition

We call the hyperplane

$$\mathbf{w}_0 \cdot \mathbf{x} + b_0 = 0 \tag{14.1}$$

optimal hyperplane

optimal if it separates the training data

$$(\mathbf{x}_1, y_1), ..., (\mathbf{x}_\ell, y_\ell), \quad \mathbf{x} \in \mathbb{R}^m,\ y \in \{-1, 1\} \tag{14.2}$$

and if the margin between the hyperplane and the closest training vector is maximal. This means that the optimal hyperplane has to satisfy the inequalities

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 \quad i = 1, ..., \ell \tag{14.3}$$

and has to minimize the functional

$$R(\mathbf{w}) = \mathbf{w} \cdot \mathbf{w}. \tag{14.4}$$

dual formulation

This quadratic optimization problem can be solved in the dual space of Lagrange multipliers. One constructs the Lagrangian

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w} \cdot \mathbf{w} - \sum_{i=1}^{\ell} \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1] \tag{14.5}$$

and finds its saddle point: the point that minimizes this functional with respect to $\mathbf{w}$ and b and maximizes it with respect to

$$\alpha_i \geq 0 \text{ for all } i = 1, ..., \ell. \tag{14.6}$$

Minimization over $\mathbf{w}$ defines the equation

$$\mathbf{w} = \sum_{i=1}^{\ell} \alpha_i y_i \mathbf{x}_i \tag{14.7}$$

and minimization over b defines the equation

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0. \tag{14.8}$$

Substituting (14.7) back into the Lagrangian (14.5) and taking into account (14.8), we obtain the functional

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j, \tag{14.9}$$

which we have to maximize with respect to parameters $\boldsymbol{\alpha}$ satisfying two constraints: equality constraint (14.8) and positivity constraints (14.6). The optimal solution $\boldsymbol{\alpha}^0 = (\alpha_1^0, \ldots, \alpha_\ell^0)$ specifies the coefficients for the optimal hyperplane

$$\mathbf{w}_0 = \sum_{i=1}^{\ell} \alpha_i^0 y_i \mathbf{x}_i. \tag{14.10}$$

decision function

Therefore the optimal hyperplane is

$$\sum_{i=1}^{\ell} \alpha_i^0 y_i \mathbf{x}_i \cdot \mathbf{x} + b_0 = 0, \tag{14.11}$$

where b_0 is chosen to maximize the margin. It is important to note that the optimal solution satisfies the Kuhn-Tucker conditions

$$\alpha_i^0 [y_i (\mathbf{w}_0 \cdot \mathbf{x}_i + b_0) - 1] = 0. \tag{14.12}$$

From these conditions it follows that if the expansion of vector $\mathbf{w}_0$ uses vector $\mathbf{x}_i$ with non-zero weight α_i^0 then the following equality must hold

$$y_i (\mathbf{w}_0 \cdot \mathbf{x}_i + b_0) = 1. \tag{14.13}$$

Vectors $\mathbf{x}_i$ that satisfy this equality are called *support vectors*.

margin

Note that the norm of vector $\mathbf{w}_0$ defines the margin ρ between optimal separating hyperplane and the support vectors

$$\rho = \frac{1}{||\mathbf{w}_0||}. \tag{14.14}$$

Therefore taking into account (14.8) and (14.13) we obtain

$$\frac{1}{\rho^2} = \mathbf{w}_0 \cdot \mathbf{w}_0 = \sum_{i=1}^{\ell} y_i \alpha_i^0 \mathbf{w}_0 \cdot \mathbf{x}_i = \sum_{i=1}^{\ell} y_i \alpha_i^0 (\mathbf{w}_0 \cdot \mathbf{x}_i + b_0) = \sum_{i=1}^{\ell} \alpha_i^0 \tag{14.15}$$

where ρ is the margin for the optimal separating hyperplane.

non-separable case

In the non-separable case we introduce slack variables ξ_i and minimize the functional

$$R(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \mathbf{w} \cdot \mathbf{w} + C \sum_{i=1}^{\ell} \xi_i \tag{14.16}$$

subject to constraints

$$y_i (\mathbf{w}_0 \cdot \mathbf{x}_i + b_0) \geq 1 - \xi_i \text{ where } \xi_i \geq 0. \tag{14.17}$$

When the constant C is sufficiently large and the data is separable, the solution of this optimization problem coincides with the one obtained for the separable case.

To solve this quadratic optimization problem for the non-separable case, we consider the Lagrangian

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w} \cdot \mathbf{w} - \sum_{i=1}^{\ell} \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i] + C \sum_{i=1}^{\ell} \xi_i - \sum_{i=1}^{\ell} \nu_i \xi_i, \quad (14.18)$$

which we minimize with respect to $\mathbf{w}$, b and ξ_i and maximize with respect to the Lagrange multipliers $\alpha_i \geq 0$ and $\nu_i \geq 0$.

The result of minimization over $\mathbf{w}$ and b leads to the conditions (14.7) and (14.8) and result of minimization over ξ_i gives the new condition

$$\alpha_i + \nu_i = C. \quad (14.19)$$

Taking into account that $\nu_i \geq 0$, we obtain

$$0 \leq \alpha_i \leq C. \quad (14.20)$$

Substituting (14.7) into the Lagrangian, we obtain that in order to find the optimal hyperplane, one has to maximize the functional (14.9), subject to constraints (14.8) and (14.20).

The box constraints (14.20) (instead of the positivity constraints (14.6)) entail the difference in the methods for constructing optimal hyperplanes in the non-separable case and in the separable case respectively. For the non-separable case, the Kuhn-Tucker conditions

$$\alpha_i^0 [y_i(\mathbf{w}_0 \cdot \mathbf{x}_i + b_0) - 1 + \xi_i] = 0 \text{ and } \nu_i \xi_i = 0 \quad (14.21)$$

must be satisfied. Vectors $\mathbf{x}_i$ that correspond to nonzero α_i^0 are referred as support vectors. For support vectors the equalities

$$y_i(\mathbf{w}_0 \cdot \mathbf{x}_i + b_0) = 1 - \xi_i \quad (14.22)$$

hold. From conditions (14.21) and (14.19) it follows that if $\xi_i > 0$, then $\nu_i = 0$ and therefore $\alpha_i = C$.

category of a support vector

We will distinguish between two types of support vectors: support vectors for which $0 < \alpha_i^0 < C$ and support vectors for which $\alpha_i^0 = C$. To simplify notations we sort the support vectors such that the first n^* support vectors belong to the first category (with $0 < \alpha_i < C$) and the next $m = n - n^*$ support vectors belong to the second category (with $\alpha_i = C$).

non linear SVM

When constructing SVMs one usually maps the input vectors $\mathbf{x} \in X$ into a high dimensional (even infinite dimensional) feature space $\phi(\mathbf{x}) \in \mathcal{F}$ where one constructs the optimal separating hyperplane. Note that both the optimal hyperplane (14.11) and the target functional (14.9) that has to be maximized to find the optimal hyperplane depend on the inner product between two vectors rather than on input vectors explicitly. Therefore one can use the general representation of inner product in order to calculate it. It is known that the inner product between

two vectors $\phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2)$ has the following general representation

$$\phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) = k(\mathbf{x}_1, \mathbf{x}_2), \tag{14.23}$$

where $k(\mathbf{x}_1, \mathbf{x}_2)$ is a kernel function that satisfies the Mercer conditions (symmetric positive definite function). The form of kernel function $k(\mathbf{x}_1, \mathbf{x}_2)$ depends on the type of mapping of the input vectors $\phi(\mathbf{x})$. In order to construct the optimal hyperplane in feature space, it is sufficient to use a kernel function instead of inner product in expressions (14.9) and (14.11).

Further we consider bounds in the input space X. However all results are true for any mapping ϕ. To obtain the corresponding results in a feature space one uses the representation of the inner product in feature space $k(\mathbf{x}, \mathbf{x}_i)$ instead of the inner product $\mathbf{x} \cdot \mathbf{x}_i$.

14.3 The Leave-one-out Procedure

The bounds introduced in this chapter are derived from the leave-one-out cross-validation estimate. This procedure is usually used to estimate the probability of test error of a learning algorithm.

Suppose that using training data of size ℓ one tries simultaneously to estimate a decision rule and evaluate the quality of this decision rule. Using training data, one constructs a decision rule. Then one uses the same training data to evaluate the quality of the obtained rule based on the *leave-one-out* procedure: one removes from the training data one element (say $(\mathbf{x}_p, y_p)$), constructs the decision rule on the basis of the remaining training data and then tests the removed element. In this fashion one tests all ℓ elements of the training data (using ℓ different decision rules). Let us denote the number of errors in the leave-one-out procedure by $\mathcal{L}(\mathbf{x}_1, y_1, ..., \mathbf{x}_\ell, y_\ell)$. Luntz and Brailovsky proved the following lemma:

leave-one-out procedure

Lemma 14.1 Luntz and Brailovsky [1969]
The leave-one-out procedure gives an almost unbiased estimate of the probability of test error

$$Ep_{error}^{\ell-1} = E\left(\frac{\mathcal{L}(\mathbf{x}_1, y_1, ..., \mathbf{x}_\ell, y_\ell)}{\ell}\right), \tag{14.24}$$

where $p_{error}^{\ell-1}$ is the probability of test error for the machine trained on a sample of size $\ell - 1$.

"Almost" in the above lemma refers to the fact the probability of test error is for samples of size $\ell - 1$ instead of ℓ.

Remark 14.2
For SVMs one needs to conduct the leave-one-out procedure only for support vectors: non support vectors will be recognized correctly since removing a point which is not a support vector does not change the decision function.

In Section 14.5, we introduce upper bounds on the number of errors made by the leave-one-out procedure. For this purpose we need to introduce a new concept, called the *span of support vectors*.

14.4 Span of the Set of Support Vectors

Let us first consider the separable case. Suppose that

$$(\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n) \tag{14.25}$$

is a set of support vectors and

$$\boldsymbol{\alpha}^0 = (\alpha_1^0, ..., \alpha_n^0) \tag{14.26}$$

is the vector of Lagrange multipliers for the optimal hyperplane.

For any fixed support vector $\mathbf{x}_p$ we define the set Λ_p as a constrained linear combinations of the points $\{\mathbf{x}_i\}_{i \neq p}$:

$$\Lambda_p = \left\{ \sum_{i=1,\ i\neq p}^{n} \lambda_i \mathbf{x}_i \ : \ \sum_{i=1,\ i\neq p}^{n} \lambda_i = 1, \text{ and } \forall i \neq p, \ \ \alpha_i^0 + y_i y_p \alpha_p^0 \lambda_i \geq 0 \right\} \tag{14.27}$$

Note that λ_i can be less than 0.

We also define the quantity S_p, which we call the *span* of the support vector $\mathbf{x}_p$
span as the distance between $\mathbf{x}_p$ and this set (see figure 14.1)

$$S_p^2 = d^2(\mathbf{x}_p, \Lambda_p) = \min_{\mathbf{x} \in \Lambda_p} (\mathbf{x}_p - \mathbf{x})^2, \tag{14.28}$$

As shown in Figure 14.2, it can happen that $\mathbf{x}_p \in \Lambda_p$, which implies $S_p = d(\mathbf{x}_p, \Lambda_p) = 0$. Intuitively, for smaller $S_p = d(\mathbf{x}_p, \Lambda_p)$ the leave-one-out procedure is less likely to make an error on the vector $\mathbf{x}_p$. Indeed, we will prove (see Lemma 14.5)) that if $S_p < 1/(D\alpha_p^0)$ (D is the diameter of the smallest sphere containing the training points), then the leave-one-out procedure classifies $\mathbf{x}_p$ correctly. By setting $\lambda_p = -1$, we can rewrite S_p as:

$$S_p^2 = \min \left\{ \left(\sum_{i=1}^{n} \lambda_i \mathbf{x}_i \right)^2 \ : \ \ \lambda_p = -1, \ \sum_{i=1}^{n} \lambda_i = 0, \ \alpha_i^0 + y_i y_p \alpha_p^0 \lambda_i \geq 0 \right\} \tag{14.29}$$

The maximal value of S_p is called the S-span

$$S = \max\{d(\mathbf{x}_1, \Lambda_1), ..., d(\mathbf{x}_n, \Lambda_n)\} = \max_p S_p. \tag{14.30}$$

We will prove (cf. Lemma 14.3 below) that $S_p \leq D_{SV}$. Therefore,

$$S \leq D_{SV}. \tag{14.31}$$

Depending on $\boldsymbol{\alpha}^0 = (\alpha_1^0, ..., \alpha_n^0)$ the value of the span S can be much less than diameter D_{SV} of the support vectors. Indeed, in the example of Figure 14.2, $d(\mathbf{x}_1, \Lambda_1) = 0$ and by symmetry, $d(\mathbf{x}_i, \Lambda_i) = 0$, for all i and we have $S = 0$.

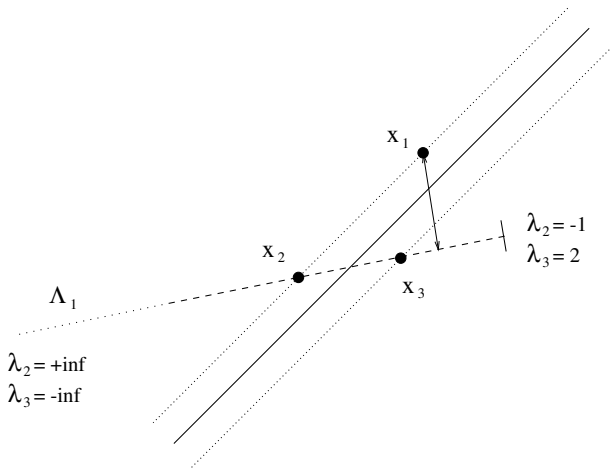


Figure 14.1 Consider the 2D example above: 3 support vectors with $\alpha_1 = \alpha_2 = \alpha_3/2$. The set Λ_1 is the semi-opened dashed line: $\Lambda_1 = \{\lambda_2\mathbf{x}_2 + \lambda_3\mathbf{x}_3,\ \lambda_2 + \lambda_3 = 1,\ \lambda_2 \geq -1,\ \lambda_3 \leq 2\}$.

non-separable case

Now we generalize the span concept for the non-separable case. In the non-separable case we distinguish between two categories of support vectors: the support vectors for which

$$0 < \alpha_i < C \text{ for all } 1 \leq i \leq n^* \tag{14.32}$$

and the support vectors for which

$$\alpha_j = C \text{ for all } = n^* + 1 \leq i \leq n. \tag{14.33}$$

We define the span of support vectors using support vectors of the first category. That means we consider the value $S_p = d(\mathbf{x}_p, \Lambda_p)$ where

$$\Lambda_p = \left\{ \sum_{i=1,\ i\neq p}^{n^*} \lambda_i \mathbf{x}_i \ : \ \sum_{i=1,\ i\neq p}^{n^*} \lambda_i = 1,\ \forall i \neq p \ \ 0 \leq \alpha_i^0 + y_i y_p \alpha_p^0 \lambda_i \leq C \right\} \tag{14.34}$$

The differences in the definition of the span for the separable and the non-separable case are that in the non-separable case we ignore the support vectors of the second category and add an upper bound C in the constraints on λ_i.

Therefore in the non-separable case the value of the span of support vectors depends on the value of C. It is not obvious that the set Λ_p is not empty. It is proven in the following lemma.

Lemma 14.3

bound on the span

Both in the separable and non-separable case, the set Λ_p is not empty. Moreover $S_p = d(\mathbf{x}_p, \Lambda_p) \leq D_{SV}$

The proof can be found in the Appendix.

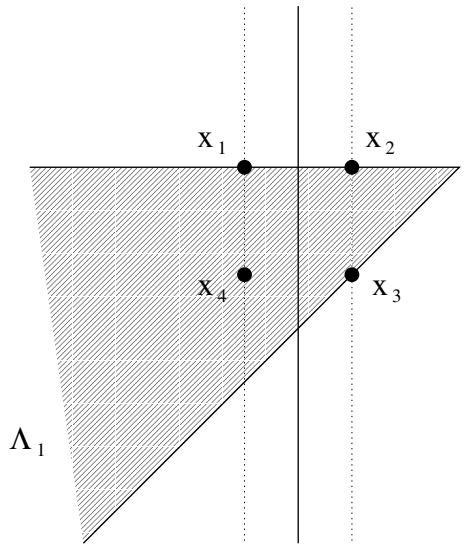


Figure 14.2 In this example, we have $\mathbf{x}_1 \in \Lambda_1$ and therefore $d(\mathbf{x}_1, \Lambda_1) = 0$. The set Λ_1 has been computed using $\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4$

Remark 14.4
From lemma 14.3, we conclude (as in the separable case) that

$$S \leq D_{SV}, \tag{14.35}$$

where D_{SV} is the diameter of the smallest sphere containing the support vectors of the first category.

14.5 The Bounds

The generalization ability of SVMs can be explained by their capacity control. Indeed, the VC dimension of hyperplanes with margin ρ is less than $D^2/4\rho^2$, where D is the diameter of the smallest sphere containing the training points [Vapnik, 1995]. This is the theoretical idea motivating the maximization of the margin.

This section presents new bounds on the generalization ability of SVMs. The major improvement lies in the fact that the bounds will depend on the span of the support vectors, which gives tighter bounds that ones depending on the diameter of the training points.

Let us first introduce our fundamental result:

Lemma 14.5
If in the leave-one-out procedure a support vector $\mathbf{x}_p$ corresponding to $0 < \alpha_p < C$ is recognized incorrectly, then the inequality

$$\alpha_p^0 S_p \max(D, 1/\sqrt{C}) \geq 1 \tag{14.36}$$

holds true.

The proof can be found in the appendix. The previous lemma leads us to the following theorem for the separable case:

Theorem 14.6
Suppose that a SVM separates training data of size ℓ without error. Then the expectation of the probability of error $p_{error}^{\ell-1}$ for the SVM trained on the training data of size $\ell - 1$ has the bound

$$Ep_{error}^{\ell-1} \leq E\left(\frac{SD}{\ell\rho^2}\right), \tag{14.37}$$

where the values of span of support vectors S, diameter of the smallest sphere containing the training points D, and the margin ρ are considered for training sets of size ℓ.

Proof Let us prove that the number of errors made by the leave-one-out procedure is bounded by $\frac{SD}{\rho^2}$. Taking the expectation and using lemma 14.1 will prove the theorem.

Consider a support vector $\mathbf{x}_p$ incorrectly classified by the leave-one-out procedure. Then Lemma 14.5 gives $\alpha_p^0 S_p D \geq 1$ (we consider the separable case and $C = \infty$) and

$$\alpha_p^0 \geq \frac{1}{SD} \tag{14.38}$$

holds true. Now let us sum the left and right hand sides of this inequality over all support vectors where the leave-one-out procedure commits an error

$$\frac{\mathcal{L}(\mathbf{x}_1, y_1, ..., \mathbf{x}_\ell, y_\ell)}{SD} \leq \sum_* \alpha_i^0. \tag{14.39}$$

Here $\sum_*$ indicates that the sum is taken only over support vectors where the leave-one-out procedure makes an error. From this inequality we have

$$\frac{\mathcal{L}(\mathbf{x}_1, y_1, ..., \mathbf{x}_\ell, y_\ell)}{\ell SD} \leq \frac{1}{\ell}\sum_{i=1}^{n} \alpha_i^0. \tag{14.40}$$

Therefore we have (using (14.15))

$$\frac{\mathcal{L}(\mathbf{x}_1, y_1, ..., \mathbf{x}_\ell, y_\ell)}{\ell} \leq \frac{SD\sum_{i=1}^{n} \alpha_i^0}{\ell} = \frac{SD}{\ell\rho^2}. \tag{14.41}$$

Taking the expectation over both sides of the inequality and using the Luntz and Brailovsky Lemma we prove the theorem. ■

For the non-separable case the following theorem is true.

Theorem 14.7
The expectation of the probability of error $p_{error}^{\ell-1}$ for a SVM trained on the training data of size $\ell - 1$ has the bound

$$Ep_{error}^{\ell-1} \leq E\left(\frac{S\max(D, 1/\sqrt{C})\sum_{i=1}^{n^*}\alpha_i^0 + m}{\ell}\right), \tag{14.42}$$

where the sum is taken only over α_i corresponding to support vectors of the first category (for which $0 < \alpha_i < C$) and m is the number of support vectors of the second category (for which $\alpha_i = C$). The values of the span of support vectors S, diameter of the smallest sphere containing the training points D, and the Lagrange multipliers $\alpha^0 = (\alpha_1^0, ..., \alpha_n^0)$ are considered for training sets of size ℓ.

Proof The proof of this theorem is similar to the proof of theorem 14.6. We consider all support vectors of the second category (corresponding to $\alpha_j = C$) as an error. For the first category of support vectors we estimate the number $\mathcal{L}^*(\mathbf{x}_1, y_1, ..., \mathbf{x}_\ell, y_\ell)$ of errors in the leave-one-out procedure using Lemma 14.5 as in the proof of Theorem 14.6. We obtain

$$\begin{aligned}\frac{\mathcal{L}(\mathbf{x}_1, y_1, ..., \mathbf{x}_\ell, y_\ell)}{\ell} &\leq \frac{\mathcal{L}^*(\mathbf{x}_1, y_1, ..., \mathbf{x}_\ell, y_\ell) + m}{\ell} \\ &\leq \frac{S\max(D, 1/\sqrt{C})\sum^*\alpha_i + m}{\ell}\end{aligned}$$

Taking the expectation over both sides of the inequality and using the Luntz and Brailovsky Lemma we prove the theorem. ■

Note that in the case when $m = 0$ (separable case), the equality (14.15) holds true. In this case (provided that C is large enough) the bounds obtained in these two theorems coincide.

Note that in Theorems 14.6 and 14.7, it is possible using inequality (14.35) to bound the value of the span S by the diameter of the smallest sphere containing the support vectors D_{SV}. But, as pointed out by the experiments (see Section 14.6), this would lead to looser bounds as the span can be much less than the diameter.

14.5.1 Extension

In the proof of Lemma 14.5, it appears that the diameter of the training points D can be replaced by the span of the support vectors *after* the leave-one-out procedure. But since the set of support vectors after the leave-one-out procedure is unknown, we bounded this unknown span by D. Nevertheless this remark motivated us to analyze the case where the set of support vectors remains the same during the leave-one-out procedure.

In this situation, we are allowed to replace D by S in lemma 14.5 and more precisely, the following theorem is true.

Theorem 14.8
If the sets of support vectors of first and second categories remain the same during the leave-one-out procedure, then for any support vector $\mathbf{x}_p$, the following equality holds:

$$y_p(f^0(\mathbf{x}_p) - f^p(\mathbf{x}_p)) = \alpha_p^0 S_p^2 \tag{14.43}$$

where f^0 and f^p are the decision function given by the SVM trained respectively on the whole training set and after the point $\mathbf{x}_p$ has been removed.

The proof can be found in the appendix.

The assumption that the set of support vectors does not change during the leave-one-out procedure is not satisfied in most cases. Nevertheless, the proportion of points which violate this assumption is usually small compared to the number of support vectors. In this case Theorem 14.8 provides a good approximation of the result of the leave-one-out procedure, as pointed out by the experiments (see Section 14.6, Figure 14.4).

Note that Theorem 14.8 is stronger than lemma 14.5 for three reasons: the term $S_p \max(D, 1/\sqrt{C})$ becomes S_p^2, the inequality turns out to be an equality and the result is valid for any support vector. The previous theorem enables us to compute the number of errors made by the leave-one-out procedure:

Corollary 14.9
Under the assumption of Theorem 14.8, the test error prediction given by the leave-one-out procedure is

span-rule

$$t_\ell = \frac{1}{\ell}\mathcal{L}(\mathbf{x}_1, y_1, ..., \mathbf{x}_\ell, y_\ell) = \frac{1}{\ell} Card\{p : \ \alpha_p^0 S_p^2 \geq y_p f^0(\mathbf{x}_p)\} \tag{14.44}$$

14.6 Experiments

The previous bounds on the generalization ability of Support Vector Machines involved the diameter of the smallest sphere enclosing the training points [Vapnik, 1995]. We have shown (cf inequality (14.35)) that the span S is always smaller than this diameter, but to appreciate the gain, we conducted some experiments.

comparison span - diameter

First we compare the diameter of the smallest sphere enclosing the training points, the one enclosing the support vectors and the span of the support vectors using the postal database. This dataset consists of 7291 handwritten digits of size 16x16 with a test set of 2007 examples. Following Schölkopf et al. [1999], we split the training set in 23 subsets of 317 training examples. Our task is to separate digits 0 to 4 from 5 to 9. Error bars in Figure 14.3 are standard deviations over the 23 trials. The diameters and the span in Figure 14.3 are plotted for different values of σ, the width of the RBF kernel we used:

$$k(\mathbf{x}, \mathbf{y}) = e^{-\frac{||\mathbf{x}-\mathbf{y}||^2}{2\sigma^2}}. \tag{14.45}$$

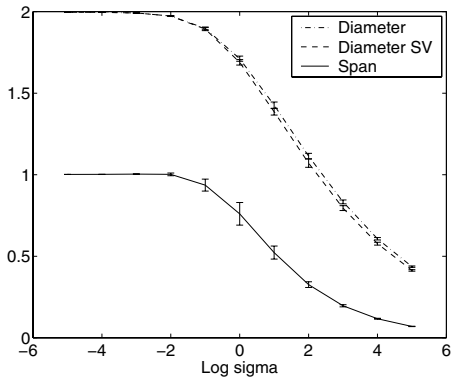


Figure 14.3 Comparison of D, D_{SV} and S

In this example, the span is up to 6 times smaller than the diameter.

Now we would like to use the span for predicting accurately the test error. This would enable us to perform efficient model selection, i.e., choosing the optimal values of parameters in SVMs (the width of the RBF kernel σ or the constant C, for instance).

Note that the span S is defined as a maximum $S = \max_p S_p$ and therefore taking into account the different values S_p should provide a more accurate estimation of the generalization error than the span S only. Therefore, we used the *span-rule* (14.44) in Corollary 1 to predict the test error.

Our experiments have been carried out on two databases: a separable one, the postal database, described above and a noisy one, the breast-cancer database.[1] The latter has been split randomly 100 times into a training set containing 200 examples and a test set containing 77 examples.

model selection

Figure 14.4 a compares the test error and its prediction given by the span-rule (14.44) for different values of the width σ of the RBF kernel on the postal database. Figure 14.4b plots the same functions for different values of C on the breast-cancer database. The prediction is very accurate and the curves are almost identical.

The computation of the span-rule (14.44) involves computing the span S_p (14.29) for every support vector. Note, however, that we are interested in the inequality $S_p^2 \leq y_p f(\mathbf{x}_p)/\alpha_p^0$, rather than the exact value of the span S_p. Therefore, if while minimizing $S_p = d(\mathbf{x}_p, \Lambda_p)$ we find a point $\mathbf{x}^* \in \Lambda_p$ such that $d(\mathbf{x}_p, \mathbf{x}^*)^2 \leq y_p f(\mathbf{x}_p)/\alpha_p^0$, we can stop the minimization because this point will be correctly classified by the leave-one-out procedure.

Figure 14.5 compares the time required to (a) train the SVM on the postal database, (b) compute the estimate of the leave-one-out procedure given by the span-rule (14.44) and (c) compute exactly the leave-one-out procedure. In order to have a fair comparison, we optimized the computation of the leave-one-out

computation time

1. Available from `http://svm.first.gmd.de/~raetsch/data/breast-cancer`

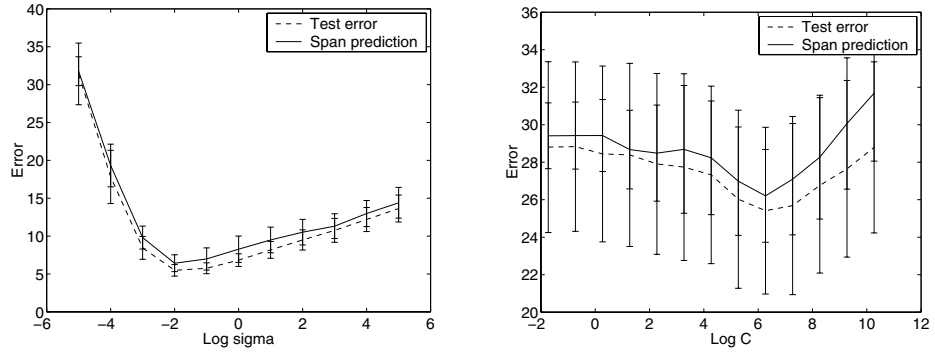


Figure 14.4 Test error and its prediction using the span-rule (14.44) (left: choice of σ in the postal database, right: choice of C in the breast-cancer database).

procedure in the following way: for every support vector $\mathbf{x}_p$, we take as starting point for the minimization (14.9) involved to compute f^p (the decision function after having removed the point $\mathbf{x}_p$), the solution given by f^0 on the whole training set. The reason is that f^0 and f^p are usually "close."

The results show that the time required to compute the span is not prohibitive and is very attractive compared to the leave-one-out procedure.

14.7 Conclusion

In this chapter, we have shown that the generalization ability of support vector machines depends on a more complicated geometrical concept than the margin only. A direct application of the concept of span is the selection of the optimal parameters of the SVM since the span enables to get an accurate prediction of the test error.

Similarly to Chapter 15, the concept of the span also leads to new learning algorithms involving the minimization of the number of errors made by the leave-one-out procedure.

Acknowledgments

The authors would like to thank Léon Bottou, Patrick Haffner and Yann LeCun and Sayan Mukherjee for very helpful discussions.

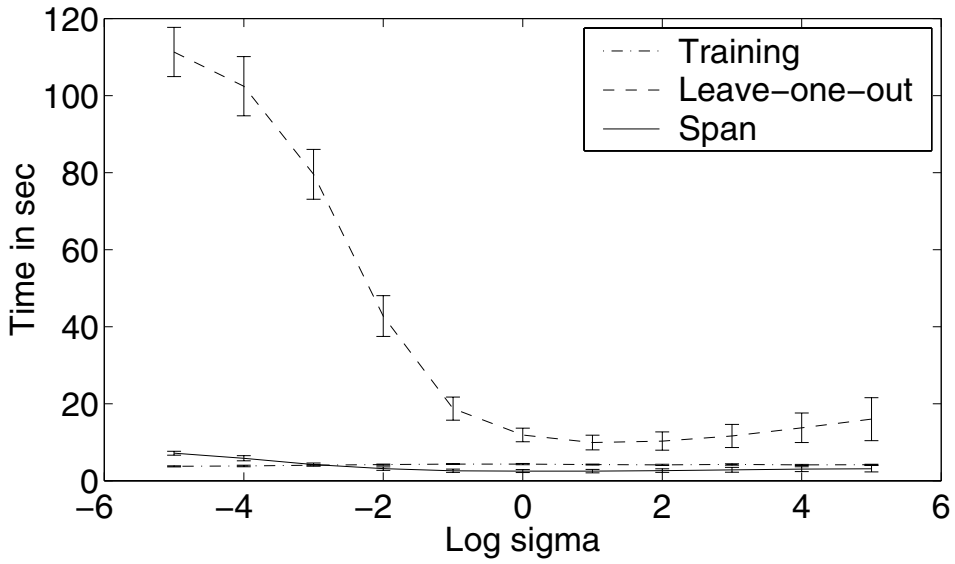


Figure 14.5 Comparison of time required for SVM training, computation of span and leave-one-out on the postal database

14.8 Appendix: Proofs

Lemma 14.3. We will prove this result for the non-separable case. The result is also valid for the separable case since it can be seen as a particular case of the non-separable one with C large enough. Let us define Λ_p^+ as the subset of Λ_p with additional constraints $\lambda_i \geq 0$:

$$\Lambda_p^+ = \{ \sum_{i=1,\, i \neq p}^{n} \lambda_i \mathbf{x}_i \in \Lambda_p \ : \ \lambda_i \geq 0 \ i \neq p \}. \tag{14.46}$$

We shall prove that $\Lambda_p^+ \neq \emptyset$ by proving that a vector λ of the following form exists:

$$\lambda_j = 0 \text{ for all } n^* + 1 \leq j \leq n \tag{14.47}$$

$$\lambda_i = \mu \frac{C - \alpha_i^0}{\alpha_p^0}, \quad y_i = y_p, \quad i \neq p, \quad i = 1, ..., n^* \tag{14.48}$$

$$\lambda_i = \mu \frac{\alpha_i^0}{\alpha_p^0}, \quad y_i \neq y_p, \quad i = 1, ..., n^* \tag{14.49}$$

$$0 \leq \mu \leq 1 \tag{14.50}$$

$$\sum_{i=1}^{n} \lambda_i = 1 \tag{14.51}$$

It is straightforward to check that if such a vector λ exists, then $\sum \lambda_i \mathbf{x}_i \in \Lambda_p^+$ and therefore $\Lambda_p^+ \neq \emptyset$. Since $\Lambda_p^+ \subset \Lambda_p$, we will have $\Lambda \neq \emptyset$.

Taking into account equations (14.48) and (14.49), we can rewrite constraint (14.51) as follows:

$$1 = \frac{\mu}{\alpha_p^0} \left(\sum_{\substack{i=1,\ i\neq p \\ y_i = y_p}}^{n^*} (C - \alpha_i^0) + \sum_{\substack{i=1 \\ y_i \neq y_p}}^{n^*} \alpha_i^0 \right) \tag{14.52}$$

We need to show that the value of μ given by equation (14.52) satisfies constraint (14.50). For this purpose, let us define Δ as:

$$\Delta = \sum_{i/\ y_i = y_p}^{n^*} (C - \alpha_i^0) + \sum_{i/\ y_i \neq y_p}^{n^*} \alpha_i^0 \tag{14.53}$$

$$= -y_p \sum_{i=1}^{n^*} y_i \alpha_i^0 + \sum_{i/\ y_i = y_p}^{n^*} C \tag{14.54}$$

Now, note that

$$\sum_{i=1}^{n} y_i \alpha_i^0 = \sum_{i=1}^{n^*} y_i \alpha_i^0 + C \sum_{i=n^*+1}^{n} y_i = 0. \tag{14.55}$$

Combining equations (14.54) and (14.55) we get

$$\begin{aligned} \Delta &= C y_p \sum_{i=n^*+1}^{n} y_i + \sum_{i/\ y_i = y_p}^{n^*} C \\ &= Ck, \end{aligned}$$

where k is an integer. Since equation (14.53) gives $\Delta > 0$, we have finally

$$\Delta \geq C. \tag{14.56}$$

Let us rewrite equation (14.52) as:

$$1 = \frac{\mu}{\alpha_p^0}(\Delta - (C - \alpha_p^0)). \tag{14.57}$$

We obtain

$$\mu = \frac{\alpha_p^0}{\Delta - (C - \alpha_p^0)} \tag{14.58}$$

Taking into account inequality (14.56), we finally get $0 \leq \mu \leq 1$. Thus, constraint (14.50) is fulfilled and Λ_p^+ is not empty. Now note that the set Λ_p^+ is included in

the convex hull of $\{\mathbf{x}_i\}_{i\neq p}$ and since $\Lambda_p^+ \neq \emptyset$, we obtain

$$d(\mathbf{x}_p, \Lambda_p^+) \leq D_{SV}, \tag{14.59}$$

where D_{SV} is the diameter of the smallest ball containing the support vectors of the first category. Since $\Lambda_p^+ \subset \Lambda_p$ we finally get

$$S_p = d(\mathbf{x}_p, \Lambda_p) \leq d(\mathbf{x}_p, \Lambda_p^+) \leq D_{SV}. \tag{14.60}$$

■

Lemma 14.5. Let us first consider the separable case. Suppose that our training set $\{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$ is ordered such that the support vectors are the first n training points. The non-zero Lagrange multipliers associated with these support vectors are

$$\alpha_1^0, ..., \alpha_n^0 \tag{14.61}$$

In other words, the vector $\boldsymbol{\alpha}^0 = (\alpha_1^0, \ldots, \alpha_n^0, 0, \ldots, 0)$ maximizes the functional

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{14.62}$$

subject to the constraints

$$\boldsymbol{\alpha} \geq 0, \tag{14.63}$$

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0. \tag{14.64}$$

Let us consider the result of the leave-one-out procedure on the support vector $\mathbf{x}_p$. This means that we maximized functional (14.62) subject to the constraints (14.63), (14.64) and the additional constraint

$$\alpha_p = 0, \tag{14.65}$$

and obtained the solution

$$\boldsymbol{\alpha}^p = (\alpha_1^p, ..., \alpha_\ell^p). \tag{14.66}$$

Using this solution we construct the separating hyperplane

$$\mathbf{w}_p \cdot \mathbf{x} + b_p = 0, \tag{14.67}$$

where

$$\mathbf{w}_p = \sum_{i=1}^{\ell} \alpha_i^p y_i \mathbf{x}_i. \tag{14.68}$$

We would like to prove that if this hyperplane classifies the vector $\mathbf{x}_p$ incorrectly:

$$y_p(\mathbf{w}_p \cdot \mathbf{x}_p + b_p) < 0 \tag{14.69}$$

then

$$\alpha_p^0 \geq \frac{1}{S_p D}. \tag{14.70}$$

Since $\boldsymbol{\alpha}^p$ maximizes (14.62) under constraints (14.63), (14.64) and (14.65), the following inequality holds true

$$W(\boldsymbol{\alpha}^p) \geq W(\boldsymbol{\alpha}^0 - \boldsymbol{\delta}), \tag{14.71}$$

where the vector $\boldsymbol{\delta} = (\delta_1, ..., \delta_n)$ satisfies the following conditions

$$\delta_p = \alpha_p^0, \tag{14.72}$$

$$\alpha_0 - \delta \geq 0, \tag{14.73}$$

$$\sum_{i=1}^{n} \delta_i y_i = 0. \tag{14.74}$$

$$\delta_i = 0, \quad i > n \tag{14.75}$$

From inequality (14.71) we obtain

$$W(\boldsymbol{\alpha}^0) - W(\boldsymbol{\alpha}^p) \leq W(\boldsymbol{\alpha}^0) - W(\boldsymbol{\alpha}^0 - \boldsymbol{\delta}). \tag{14.76}$$

Since $\boldsymbol{\alpha}^0$ maximizes (14.62) under the constraints (14.63) and (14.64), the following inequality holds true

$$W(\boldsymbol{\alpha}^0) \geq W(\boldsymbol{\alpha}^p + \boldsymbol{\gamma}), \tag{14.77}$$

where $\boldsymbol{\gamma} = (\gamma_1, ..., \gamma_\ell)$ is a vector satisfying the constraints

$$\boldsymbol{\alpha}_p + \boldsymbol{\gamma} \geq 0, \tag{14.78}$$

$$\sum_{i=1}^{\ell} \gamma_i y_i = 0. \tag{14.79}$$

$$\alpha_i^p = 0 \Longrightarrow \gamma_i = 0, \quad i \neq p \tag{14.80}$$

From (14.76) and (14.77), we have

$$W(\boldsymbol{\alpha}^p + \boldsymbol{\gamma}) - W(\boldsymbol{\alpha}^p) \leq W(\boldsymbol{\alpha}_0) - W(\boldsymbol{\alpha}_0 - \boldsymbol{\delta}) \tag{14.81}$$

Let us calculate both the left hand side,I_1, and the right hand side, I_2 of inequality (14.81).

$$\begin{aligned} I_1 &= W(\boldsymbol{\alpha}^p + \boldsymbol{\gamma}) - W(\boldsymbol{\alpha}^p) \\ &= \sum_{i=1}^{\ell} (\alpha_i^p + \gamma_i) - \frac{1}{2} \sum_{i,j=1}^{\ell} (\alpha_i^p + \gamma_i)(\alpha_j^p + \gamma_j) y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\ &\quad - \sum_{i=1}^{\ell} \alpha_i^p + \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i^p \alpha_j^p y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \end{aligned}$$

$$= \sum_{i=1}^{\ell} \gamma_i - \sum_{i,j}^{\ell} \gamma_i \alpha_j^p y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j - \frac{1}{2} \sum_{i,j}^{\ell} y_i y_j \gamma_i \gamma_j \mathbf{x}_i \cdot \mathbf{x}_j$$
$$= \sum_{i=1}^{\ell} \gamma_i (1 - y_i \mathbf{w}_p \cdot \mathbf{x}_i) - \frac{1}{2} \sum_{i,j}^{\ell} y_i y_j \gamma_i \gamma_j (\mathbf{x}_i, \mathbf{x}_j)$$

Taking into account that

$$\sum_{i=1}^{\ell} \gamma_i y_i = 0 \tag{14.82}$$

we can rewrite expression

$$I_1 = \sum_{i \neq p}^{\ell} \gamma_i [1 - y_i (\mathbf{w}_p \cdot \mathbf{x}_i + b_p)] + \gamma_p [1 - y_p (\mathbf{w}_p \cdot \mathbf{x}_p + b_p)] - \frac{1}{2} \sum_{i,j}^{n} y_i y_j \gamma_i \gamma_j \mathbf{x}_i \cdot \mathbf{x}_j. \tag{14.83}$$

Since for $i \neq p$ the condition (14.80) means that either $\gamma_i = 0$ or $\mathbf{x}_i$ is a support vector of the hyperplane $\mathbf{w}_p$, the following equality holds

$$\gamma_i [y_i (\mathbf{w}_p \cdot \mathbf{x}_i + b_p) - 1] = 0. \tag{14.84}$$

We obtain

$$I_1 = \gamma_p [1 - y_p (\mathbf{w}_p \cdot \mathbf{x}_p + b_p)] - \frac{1}{2} \sum_{i,j}^{n} y_i y_j \gamma_i \gamma_j \mathbf{x}_i \cdot \mathbf{x}_j. \tag{14.85}$$

Now let us define vector γ as follows:

$$\gamma_p = \gamma_k = a, \tag{14.86}$$

$$\gamma_i = 0 \quad i \notin \{k, p\}, \tag{14.87}$$

where a is some constant and k such that $y_p \neq y_k$ and $\alpha_k^p > 0$. For this vector we obtain

$$I_1 = a[1 - y_p (\mathbf{w}_p \cdot \mathbf{x}_p + b_p)] - \frac{a^2}{2} ||\mathbf{x}_p - \mathbf{x}_k||^2$$
$$\geq a[1 - y_p (\mathbf{w}_p \cdot \mathbf{x}_p + b_p)] - \frac{a^2}{2} D^2. \tag{14.88}$$

Let us choose the value a to maximize this expression

$$a = \frac{1 - y_p (\mathbf{w}_p \cdot \mathbf{x}_p + b_p)}{D^2}. \tag{14.89}$$

Putting this expression back into (14.88) we obtain

$$I_1 \geq \frac{1}{2} \frac{(1 - y_p [(\mathbf{x}_p, \mathbf{w}_p) + b_p])^2}{D^2}. \tag{14.90}$$

Since, according to our assumption, the leave-one-out procedure commits an error at the point $\mathbf{x}_p$ (that is, the inequality (14.69) is valid), we obtain

$$I_1 \geq \frac{1}{2D^2}. \tag{14.91}$$

Now we estimate the right hand side of inequality (14.81)

$$I_2 = W(\boldsymbol{\alpha}^0) - W(\boldsymbol{\alpha}^0 - \boldsymbol{\delta}). \tag{14.92}$$

We choose $\delta_i = -y_i y_p \alpha_p^0 \lambda_i$, where λ is the vector that defines the value of $d(\mathbf{x}_p, \Lambda_p)$ in equation (14.29). We have

$$\begin{aligned} I_2 &= W(\boldsymbol{\alpha}^0) - W(\boldsymbol{\alpha}^0 - \boldsymbol{\delta}) \\ &= \sum_{i=1}^{n} \alpha_i^0 - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i^0 \alpha_j^0 y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j - \sum_{i=1}^{n} (\alpha_i^0 + y_i y_p \alpha_p^0 \lambda_i) \\ &\quad + \frac{1}{2} \sum_{i,j=1}^{n} (\alpha_i^0 + y_i y_p \alpha_p^0 \lambda_i)(\alpha_j^0 + y_j y_p \alpha_p^0 \lambda_j) y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\ &= -y_p \alpha_p^0 \sum_{i=1}^{n} y_i \lambda_i + y_p \alpha_p^0 \sum_{i,j=1}^{n} \alpha_i^0 \lambda_j y_i \mathbf{x}_i \cdot \mathbf{x}_j + \frac{1}{2} (\alpha_p^0)^2 \left(\sum_{i=1}^{n} \lambda_i \mathbf{x}_i \right)^2. \end{aligned}$$

Since $\sum_{i=1}^{n} \lambda_i = 0$ and $\mathbf{x}_i$ is a support vector, we have

$$I_2 = y_p \alpha_p^0 \sum_{i=1}^{n} \lambda_i y_i [y_i (\mathbf{w}_0 \cdot \mathbf{x}_i + b_0) - 1] + \frac{(\alpha_p^0)^2}{2} \left(\sum_{i=1}^{n} \lambda_i \mathbf{x}_i \right)^2 = \frac{(\alpha_p^0)^2}{2} S_p^2. \tag{14.93}$$

Combining (14.81), (14.91) and (14.93) we obtain

$$\alpha_p^0 S_p D \geq 1. \tag{14.94}$$

Consider now the non-separable case. The sketch of the proof is the same. There are only two differences: First, the vector $\boldsymbol{\gamma}$ needs to satisfy $\boldsymbol{\alpha}_p + \boldsymbol{\gamma} \leq C$. A very similar proof to the one of lemma 14.3 gives us the existence of $\boldsymbol{\gamma}$. The other difference lies in the choice of a in equation (14.88). The value of a which maximizes equation (14.88) is

$$a^* = \frac{1 - y_p(\mathbf{w}_p \cdot \mathbf{x}_p + b_p)}{D^2}. \tag{14.95}$$

But we need to fulfill the condition $a \leq C$. Thus, if $a^* > C$, we replace a by C in equation (14.88) and we get:

$$\begin{aligned} I_1 &\geq C[1 - y_p(\mathbf{w}_p \cdot \mathbf{x}_p + b_p)] - \frac{C^2}{2} D^2 \\ &= CD^2 \left(a^* - \frac{C}{2} \right) \geq CD^2 \frac{a^*}{2} = \frac{C}{2} [1 - y_p(\mathbf{w}_p \cdot \mathbf{x}_p + b_p)] \geq \frac{C}{2} \end{aligned}$$

The last inequality comes from (14.69). Finally, we have

$$I_1 \geq \frac{1}{2} \min \left(C, \frac{1}{D^2} \right). \tag{14.96}$$

By combining this last inequality with (14.81) and (14.93) we prove the lemma. ■

Theorem 14.8. The proof follows the proof of Lemma 14.5. Under the assumption that the set of support vectors remain the same during the leave-one-out procedure, we can take $\boldsymbol{\delta} = \boldsymbol{\gamma} = \boldsymbol{\alpha}^0 - \boldsymbol{\alpha}^p$ as $\boldsymbol{\alpha}^0 - \boldsymbol{\alpha}^P$ is a vector satisfying simultaneously the set of constraints (14.75) and (14.80). Then inequality (14.81) becomes an equality:

$$I_1 = W(\boldsymbol{\alpha}^0) - W(\boldsymbol{\alpha}^p) = I_2 \tag{14.97}$$

From inequality (14.76), it follows that

$$I_2 \leq I_2^* = W(\boldsymbol{\alpha}^0) - W(\boldsymbol{\alpha}^0 - \boldsymbol{\delta}^*), \tag{14.98}$$

where $\delta_i^* = -y_i y_p \alpha_p^0 \lambda_i$ and λ is given by the definition of the span S_p (cf equation (14.29)). The computation of I_2 and I_2^* is similar to the one involved in the proof of Lemma 14.5 (cf equation (14.93))

$$I_2^* = \frac{(\alpha_p^0)^2}{2} S_p^2 - \alpha_p^0 [y_p(\mathbf{w}_0 \cdot \mathbf{x}_p + b_0) - 1] \tag{14.99}$$

$$I_2 = \frac{(\alpha_p^0)^2}{2} \left(\sum_i \lambda_i^* \mathbf{x}_i \right)^2 - \alpha_p^0 [y_p(\mathbf{w}_0 \cdot \mathbf{x}_p + b_0) - 1], \tag{14.100}$$

where

$$\lambda_i^* = y_i \frac{\alpha_i^p - \alpha_i^0}{\alpha_p^0} \tag{14.101}$$

From (14.98), we get $\left(\sum_i \lambda_i^* \mathbf{x}_i\right)^2 \leq S_p^2$. Now note that $\sum_{i \neq p} \lambda_i^* \mathbf{x}_i \in \Lambda_p$ and by definition of S_p, $\left(\sum_i \lambda_i^* \mathbf{x}_i\right)^2 \geq S_p^2$. Finally, we have

$$\left(\sum_i \lambda_i^* \mathbf{x}_i \right)^2 = S_p^2. \tag{14.102}$$

The computation of I_1 gives (cf. equation (14.88))

$$I_1 = \alpha_p^0 [1 - y_p(\mathbf{w}_p \cdot \mathbf{x}_p + b_p)] - \frac{(\alpha_p^0)^2}{2} \left(\sum_i \lambda_i^* \mathbf{x}_i \right)^2 \tag{14.103}$$

Putting the values of I_1 and I_2 back in equation (14.97), we get

$$(\alpha_p^0)^2 \left(\sum_i \lambda_i^* \mathbf{x}_i \right)^2 = \alpha_p^0 y_p [f^0(\mathbf{x}_p) - f^p(\mathbf{x}_p)] \tag{14.104}$$

and the theorem is proven by dividing by α_p^0 and taking into account (14.102):

$$\alpha_p^0 S_p^2 = y_p [f^0(\mathbf{x}_p) - f^p(\mathbf{x}_p)] \tag{14.105}$$

■

15 Adaptive Margin Support Vector Machines

Jason Weston
Royal Holloway, University of London
Department of Computer Science,
Egham, Surrey, TW20 OEX, UK
jasonw@dcs.rbnc.ac.uk

Ralf Herbrich
Technical University of Berlin
Department of Computer Science,
Franklinstr. 28/29,
10587 Berlin, Germany
ralfh@cs.tu-berlin.de

In this chapter we present a new learning algorithm, Leave–One–Out (LOO–) SVMs and its generalization Adaptive Margin (AM–) SVMs, inspired by a recent upper bound on the leave–one–out error proved for kernel classifiers by Jaakkola and Haussler. The new approach minimizes the expression given by the bound in an attempt to minimize the leave–one–out error. This gives a convex optimization problem which constructs a sparse linear classifier in *feature space* using the kernel technique. As such the algorithm possesses many of the same properties as SVMs and Linear Programming (LP–) SVMs. These former techniques are based on the minimization of a regularized margin loss, where the margin is treated *equivalently* for each training pattern. We propose a minimization problem such that *adaptive margins* for each training pattern are utilized. Furthermore, we give bounds on the generalization error of the approach which justifies its robustness against outliers. We show experimentally that the generalization error of AM–SVMs is comparable to SVMs and LP–SVMs on benchmark datasets from the UCI repository.

15.1 Introduction

The study of classification learning has shown that algorithms which learn a *real–valued* function for classification can control their generalization error by making use of a quantity known as the *margin* (see Section 1.1.3). Based on these results, learning machines which *directly* control the margin (e.g., SVMs, LP–SVMs) have been proven to be successful in classification learning [Mason et al., 1999, Vapnik, 1998, Smola, 1998]. Moreover, it turned out to be favourable to formulate the decision functions in terms of a symmetric, positive semidefinite, and square integrable function $k(\cdot,\cdot)$ referred to as a *kernel* (see Section 1.3.2). The class of decision functions — also known as *kernel classifiers* [Smola, 1998, Jaakkola and Haussler, 1999b] — is then given by[1]

decision function

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) \qquad \boldsymbol{\alpha} \geq \mathbf{0}\,. \tag{15.1}$$

For simplicity we ignore classifiers which use an extra threshold term (cf. Eq. (1.74)).

Whilst the algorithms proposed so far are restricted to a *fixed margin* (the same constant value) at each training pattern $(\mathbf{x}_i, y_i)$, we show that *adaptive* margins can successfully be used. Moreover, it turns out that adaptive margins effectively control the complexity of the model. The chapter is structured as follows: In Section 15.2 we describe the LOO-SVM algorithm. The generalization of LOO-SVMs to control the margin adaptively, which gives AM-SVMs, is then presented in Section 15.3 and their relation to SVMs and LP–SVMs is revealed in Section 15.4. In Section 15.5 we give bounds on the generalization error of AM–SVMs which justify the use of adaptive margins as a regularizer. In Section 15.6 results of a comparison of AM–SVMs with SVMs on artificial and benchmark datasets from the UCI repository[2] are presented. Finally, in Section 15.7 we summarize the chapter and discuss further directions.

15.2 Leave–One–Out Support Vector Machines

Support Vector Machines obtain sparse solutions that yield a direct assessment of generalization: the leave-one-out error is bounded by the expected ratio of the number of non–zero coefficients α_i to the number m of training examples [Vapnik, 1995]. Jaakkola and Haussler [1999b] derive a bound on this error for a class of classifiers which includes SVMs but can be applied to non–sparse solutions. In

1. Although this class of functions is dependent on the training set, the restrictions put on $k(\cdot,\cdot)$ automatically ensure that the influence of each *new* basis function $k(\mathbf{x}_i,\cdot)$ decreases rapidly for increasing training set sizes m. Thus we can assume the existence of a *fixed* feature space (see, e.g., [Graepel et al., 1999]).
2. http://www.ics.uci.edu/mlearn/MLRepository.html.

order to motivate our reasoning we restate their result which is given by (1.86) in a more concise form.

leave–one–out bound

Theorem 15.1

For any training set of examples $\mathbf{x}_i \in R^N$ and labels $y_i \in \{\pm 1\}$, for an SVM the leave–one–out error estimate of the classifier is bounded by

$$\frac{1}{m}\sum_{i=1}^{m}\theta\left(-y_i\sum_{j\neq i}y_j\alpha_j k(\mathbf{x}_i,\mathbf{x}_j)\right). \tag{15.2}$$

where $\theta(\cdot)$ is the step function.

This bound is slightly tighter than the classical SVM leave–one–out bound. This is easy to see when one considers that all training points that have $\alpha_i = 0$ cannot be leave–one–out errors in either bound. Vapnik's bound assumes all support vectors (all training points with $\alpha_i > 0$) are errors, whereas they only contribute as errors in Equation (15.2) if

$$y_i\sum_{j\neq i}\alpha_j y_j k(\mathbf{x}_i,\mathbf{x}_j) \leq 0. \tag{15.3}$$

In practice this means the bound is tighter for less sparse solutions.

Theorem 15.1 motivates the following algorithm [Weston, 1999]: directly minimize the expression in the bound. In order to achieve this, one introduces slack variables following the standard approach of Cortes and Vapnik [1995] to give the following optimization problem:

$$\text{minimize} \quad \sum_{i=1}^{m}\xi_i^{\delta} \tag{15.4}$$

$$\text{subject to} \quad y_i\sum_{j\neq i}\alpha_j y_j k(\mathbf{x}_i,\mathbf{x}_j) \geq 1-\xi_i, \quad \text{for all } i=1,\ldots,m \tag{15.5}$$

$$\boldsymbol{\alpha}\geq\mathbf{0},\boldsymbol{\xi}\geq\mathbf{0}. \tag{15.6}$$

where one chooses a fixed constant for the margin to ensure non–zero solutions.

To make the optimization problem tractable, the smallest value for δ for which we obtain a convex objective function is $\delta = 1$. Noting also that $y_i\sum_{j\neq i}\alpha_j y_j k(\mathbf{x}_i,\mathbf{x}_j) = y_i f(\mathbf{x}_i) - \alpha_i k(\mathbf{x}_i,\mathbf{x}_i)$ we obtain the equivalent linear program:

Leave–one–out SVM

$$\text{minimize} \quad \sum_{i=1}^{m}\xi_i \tag{15.7}$$

$$\text{subject to} \quad y_i f(\mathbf{x}_i) \geq 1-\xi_i+\alpha_i k(\mathbf{x}_i,\mathbf{x}_i), \quad \text{for all } i=1,\ldots,m \tag{15.8}$$

$$\boldsymbol{\alpha}\geq\mathbf{0},\boldsymbol{\xi}\geq\mathbf{0}. \tag{15.9}$$

As in other kernel classifiers, one uses the decision rule given in Equation (15.1). Note that Theorem 15.1 is no longer valid for this learning algorithm. Nevertheless, let us study the resulting method which we call a Leave–One–Out Support Vector Machine (LOO–SVM).

regularization

Firstly, the technique appears to have no free regularization parameter.[3] This should be compared with Support Vector Machines which control the amount of regularization with the free parameter C (see Section 1.3). For SVMs, in the case of $C = \infty$ one obtains a *hard margin* classifier with no training errors. In the case of noisy or linear inseparable datasets[4] (through noise, outliers, or class overlap) one must admit some training errors (by constructing a so called *soft margin* – see Section 1.1.4). To find the best choice of training error/margin tradeoff one has to choose the appropriate value of C. In LOO–SVMs a soft margin is automatically constructed. This happens because the algorithm does not attempt to minimize the number of training errors – it minimizes the number of training points that are classified incorrectly even when they are removed from the linear combination that forms the decision rule. However, if one can classify a training point correctly when it is removed from the linear combination then it will always be classified correctly when it is placed back into the rule. This can be seen as $\alpha_i y_i k(\mathbf{x}_i, \mathbf{x}_i)$ has always the same sign as y_i, any training point is pushed further from the decision boundary by its own component of the linear combination. Note also that summing for all $j \neq i$ in the constraint (15.5) is equivalent to setting the diagonal of the kernel matrix to zero and instead summing for all j. Thus the regularization employed by

sparsity

LOO–SVMs disregards the values $k(\mathbf{x}_i, \mathbf{x}_i) = 0$ for all i.

Secondly, like Support Vector machines, the solutions can be sparse; that is, only some of the coefficients $\alpha_i,\ i = 1, \ldots, m$ are non–zero (see Section 15.6.2 for computer simulations confirming this). As the coefficient of a training point does not contribute to its leave-one-out error in constraint (15.5) the algorithm does not assign a non–zero value to the coefficient of a training point in order to correctly classify it. A training point has to be classified correctly by the training points of the same label that are close to it (in *feature space*), but the training point itself makes no contribution to its own classification.

In the next section we show how this method does in fact have an implicit regularization parameter and generalize the method to control the regularization on the set of decision functions.

15.3 Adaptive Margin SVMs

In the setting of the optimization problem (15.7)–(15.9) it is easy to see that a training point $\mathbf{x}_i$ is linearly penalized for failing to obtain a margin of $\rho_f(\mathbf{x}_i, y_i) \geq 1 + \alpha_i k(\mathbf{x}_i, \mathbf{x}_i)$. That is, the larger the contribution the training point has to the decision rule (the larger the value of α_i), the larger its margin must be. Thus,

3. As we shall see later there is an implicit regularization parameter, but it is fixed. The generalization of this problem which allows one to control this parameter gives Adaptive Margin SVMs.

4. Here we refer to linearly inseparability in *feature space*. Both SVMs and LOO–SVM Machines are essentially linear classifiers.

the algorithm controls the margin for each training point *adaptively*. From this formulation one can generalize the algorithm to control regularization through the margin loss. To make the margin at each training point a controlling variable we propose the following learning algorithm:

Adaptive Margin SVM

$$\text{minimize} \quad \sum_{i=1}^{m} \xi_i \tag{15.10}$$

$$\text{subject to} \quad y_i f(\mathbf{x}_i) \geq 1 - \xi_i + \lambda \alpha_i k(\mathbf{x}_i, \mathbf{x}_i), \quad \text{for all } i = 1, \ldots, m. \tag{15.11}$$

$$\boldsymbol{\alpha} \geq \mathbf{0}, \boldsymbol{\xi} \geq \mathbf{0}. \tag{15.12}$$

This algorithm can then be viewed in the following way (see Figure 15.1): Suppose the data lives on the surface of a hypersphere in $\mathcal{F}$, e.g., $k(\cdot, \cdot)$ is an RBF kernel given by Equation (1.73). Then $k(\mathbf{x}_i, \mathbf{x}_j)$ is the cosine of the angle between $\Phi(\mathbf{x}_i)$ and $\Phi(\mathbf{x}_j)$. As soon as a point $\Phi(\mathbf{x}_k)$ is an outlier (the cosine of the angles to points in its class are small and to points in the other class are large) α_k in Equation (15.11) has to be large in order to classify $\Phi(\mathbf{x}_k)$ correctly. Whilst SVMs and LP–SVMs use the same margin for such an outlier, they attempt to classify $\Phi(\mathbf{x}_k)$ correctly. In AM–SVMs the margin is automatically increased to $1 + \lambda \alpha_k k(\mathbf{x}_k, \mathbf{x}_k)$ for $\Phi(\mathbf{x}_k)$ and thus less attempt is made to change the decision function.

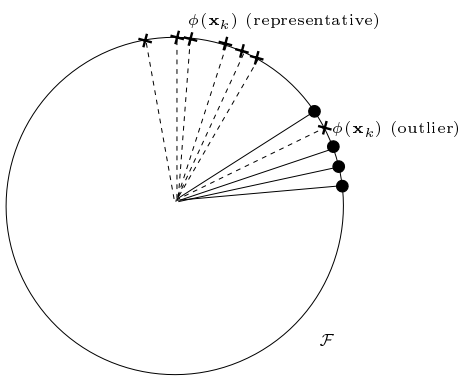


Figure 15.1 Adaptation of margins at each training pattern depending on the distance $k(\mathbf{x}_i, \mathbf{x}_j)$ in feature space $\mathcal{F}$. Note that $k(\mathbf{x}_i, \mathbf{x}_j)$ is large if the enclosed angle between data points is small. See the text for explanation.

cluster centres

Moreover, in AM–SVMs the points $\Phi(\mathbf{x}_k)$ which are representatives of clusters (centres) in feature space $\mathcal{F}$, i.e., those which have large values of the cosine of the angles to points from their class, will have non–zero α_k. In order to see this we consider two points k and k' of the same class. Let us assume that k having $\xi_k > 0$ is the centre of a cluster (in the metric induced by Φ) and k' (having $\xi_{k'} > 0$) lies

at the boundary of the cluster. Hence we subdivide the set of all points into

$$\begin{aligned}
&i \in C^+ && \xi_i = 0, y_i = y_k, i \neq k, i \neq k' \\
&i \in C^- && \xi_i = 0, y_i \neq y_k \\
&i \in I^+ && \xi_i > 0, y_i = y_k, i \neq k, i \neq k' \\
&i \in I^- && \xi_i > 0, y_i \neq y_k
\end{aligned}$$

We consider the change in $\boldsymbol{\xi}$ if we increase α_k by $\Delta > 0$ (giving $\boldsymbol{\xi}'$) and simultaneously decrease $\alpha_{k'}$ by Δ (giving $\boldsymbol{\xi}''$). From Equation (15.10)-(15.12) we know that

$$\begin{array}{llllll}
i \in C^+ & \xi_i' & = & \xi_i & \xi_i'' & \leq & \Delta k(\mathbf{x}_i, \mathbf{x}_{k'}) \\
i \in C^- & \xi_i' & \leq & \Delta k(\mathbf{x}_i, \mathbf{x}_k) & \xi_i'' & = & \xi_i \\
i \in I^+ & \xi_i' & \geq & \xi_i - \Delta k(\mathbf{x}_i, \mathbf{x}_k) & \xi_i'' & = & \xi_i + \Delta k(\mathbf{x}_i, \mathbf{x}_{k'}) \\
i \in I^- & \xi_i' & = & \xi_i + \Delta k(\mathbf{x}_i, \mathbf{x}_k) & \xi_i'' & \geq & \xi_i - \Delta k(\mathbf{x}_i, \mathbf{x}_{k'}) \\
i = k & \xi_k' & \geq & \xi_k - \Delta(1-\lambda) k(\mathbf{x}_k, \mathbf{x}_k) & \xi_k'' & = & \xi_k + \Delta k(\mathbf{x}_k, \mathbf{x}_{k'}) \\
i = k' & \xi_{k'}' & \geq & \xi_{k'} - \Delta k(\mathbf{x}_{k'}, \mathbf{x}_k) & \xi_{k'}'' & \geq & \xi_{k'} + (1-\lambda)\Delta k(\mathbf{x}_{k'}, \mathbf{x}_{k'})
\end{array}$$

Now we choose the largest Δ such that all inequalities for $i \in \{I^+, I^-, k, k'\}$ become equalities and the r.h.s for all inequalities for $i \in \{C^+, C^-\}$ equal zero. Then, the relative change in the objective function is given by

$$\frac{1}{\Delta}\sum_{i=1}^{\ell}(\xi_i' - \xi_i'') = \underbrace{\sum_{i \in I^+}\left(k(\mathbf{x}_i, \mathbf{x}_{k'}) - k(\mathbf{x}_i, \mathbf{x}_k)\right)}_{\text{change of intra–class distance}} - \underbrace{\sum_{i \in I^-}\left(k(\mathbf{x}_i, \mathbf{x}_{k'}) - k(\mathbf{x}_i, \mathbf{x}_k)\right)}_{\text{change of inter–class distance}},$$

where we assumed that $k(\mathbf{x}_k, \mathbf{x}_k) = k(\mathbf{x}_{k'}, \mathbf{x}_{k'})$. Since the cluster centres in feature space $\mathcal{F}$ minimize the intra–class distance whilst maximizing the inter–class distances it becomes apparent that their α_k will be higher. Taking into account that the maximal Δ to be considerable for this analysis is decreasing as λ increases we see that for suitably small λ AM–SVMs tend to give non–zero α's only to cluster centres in feature space $\mathcal{F}$ (see also Section 15.6 and Figure 15.4).

It is worthwhile to study the influence of λ:

- If $\lambda = 0$ no adaptation of the margins is performed. This is equivalent to minimizing training error with no regularization, i.e., approximating the expected risk $R(f)$ (1.26) with the empirical risk (1.27) (see Section 1.2).
- If $\lambda \to \infty$ the margin at each point tends to infinity ($1 + \lambda\alpha_i k(\mathbf{x}_i, \mathbf{x}_i)$) and the solution is thus to set all α's to an equal and small value. This corresponds to paying *no* attention to $R_{\text{emp}}(f)$ and is equivalent to density estimation on each class (Parzen's windows) [Parzen, 1962a].
- If $\lambda = 1$ the resulting algorithm is equivalent to LOO–SVMs.

15.4 Relationship of AM–SVMs to Other SVMs

Using the soft margin loss

$$c(\mathbf{x}, y, f(\mathbf{x})) = \max(1 - yf(\mathbf{x}), 0) \tag{15.13}$$

one can derive SVMs and LP-SVMs by choosing different regularizers. If we use the quadratic regularization functional

$$Q_{\mathrm{QP}}(f) = \|\mathbf{w}\|_2^2\,, \tag{15.14}$$

SVMs

we directly obtain the well known class of SVMs (see Section 1.3), i.e.,

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m} \xi_i + \lambda \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to} & y_i f(\mathbf{x}_i) \geq 1 - \xi_i, \quad \text{for all } i = 1, \ldots, m \\ & \boldsymbol{\alpha} \geq \mathbf{0}, \boldsymbol{\xi} \geq \mathbf{0}. \end{array} \tag{15.15}$$

Here we used

$$\mathbf{w} = \sum_{j=1}^{m} \alpha_j y_j \Phi(\mathbf{x})\,, \tag{15.16}$$

where $\Phi(\cdot)$ maps into a feature space $\mathcal{F}$ such that $(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')) = k(\mathbf{x}, \mathbf{x}')$. It is known that $Q_{\mathrm{QP}}(f)$ controls the covering number $\mathcal{N}(\cdot, F)$ of the induced loss–function class (Theorem 1.5) [Shawe-Taylor et al., 1998, Smola, 1998]. This choice of regularizer favours flat functions in feature space.

LP–SVMs

Similarly using a linear regularization functional

$$Q_{\mathrm{LP}}(f) = \sum \alpha_i \tag{15.17}$$

we obtain LP–SVMs. The corresponding minimization problem is given by[5]

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m} \xi_i + \lambda \sum_{i=1}^{m} \alpha_i \\ \text{subject to} & y_i f(\mathbf{x}_i) \geq 1 - \xi_i, \quad \text{for all } i = 1, \ldots, m \\ & \boldsymbol{\alpha} \geq \mathbf{0}, \boldsymbol{\xi} \geq \mathbf{0}. \end{array} \tag{15.18}$$

Recently it was shown that also $Q_{\mathrm{LP}}(f)$ can also be used to control the covering number of $c(\cdot, \cdot, f(\cdot))$ [Smola, 1998]. In contrast to the quadratic regularizer, $Q_{\mathrm{LP}}(f)$ favours non–smooth functions by strongly penalizing basis functions $\Phi_j(\cdot)$ with a small eigenvalue [Smola, 1998].

Comparing these algorithms to AV-SVMs, one can see all three produce a sparse kernel classifier. It is easy to see that for $\lambda = 0$ and $\lambda \to \infty$ all three algorithms revert to the same learnt function. It is only how λ stratifies the set of decision functions to form the type of regularization that differentiates the three algorithms.

5. Note, that we require $\boldsymbol{\alpha} \geq \mathbf{0}$ which allows us to omit the absolute values on the α_i's.

15.5 Theoretical Analysis

To obtain margin distribution bounds for Adaptive Margin Machines we apply the following theorem to be found in [Shawe-Taylor and Cristianini, 1999b]:

Theorem 15.2
Consider a fixed but unknown probability distribution on the input space $\mathcal{X}$ with support in the ball of radius R about the origin. Then with probability $1-\delta$ over randomly drawn training sets (X,Y) of size m for all $\rho > 0$ such that $d((\mathbf{x},y),\mathbf{w},\rho) = 0$, for some $(\mathbf{x},y) \in (X,Y)$, the generalization of a linear classifier $\mathbf{w}$ on $\mathcal{X}$ satisfying $\|\mathbf{w}\|_{\mathcal{X}} \leq 1$ is bounded from above by

$$\epsilon = \frac{2}{m}\left(\kappa \log_2\left(\frac{8em}{\kappa}\right)\log_2(32m) + \log_2\left(\frac{2m(28+\log_2(m))}{\delta}\right)\right), \tag{15.19}$$

where

$$\kappa = \left\lfloor \frac{65[(R+D)^2 + 2.25RD]}{\rho^2} \right\rfloor, \tag{15.20}$$

$$D = D(S,\mathbf{w},\rho) = \sqrt{\sum_{i=1}^{m} d_i^2}$$

$$d_i = d((\mathbf{x}_i,y),\mathbf{w},\rho) = \max\{0, \rho - y(\mathbf{w}\cdot\mathbf{x}_i)\}$$

and provided $m \geq \max\{2/\epsilon, 6\}$ and $\kappa \leq em$.

Applying the bound to AM–SVMs we can give the following theorem.

Theorem 15.3
Consider a fixed but unknown probability distribution on the feature space $\mathcal{F}$ with support in the ball of radius R about the origin. Then with probability $1-\delta$ over randomly drawn training sets (X,Y) of size m for $\boldsymbol{\alpha} \geq \mathbf{0}$ and $\boldsymbol{\xi} \geq \mathbf{0}$ which are feasible solutions of AM–SVMs such that $d((\mathbf{x},y),\mathbf{w},1) = 0$ for some $(\mathbf{x},y) \in (X,Y)$, the generalization error $R(f)$ is bounded by

$$\epsilon = \frac{2}{m}\left(\kappa \log_2\left(\frac{8em}{\kappa}\right)\log_2(32m) + \log_2\left(\frac{2m(28+\log_2(m))}{\delta}\right)\right), \tag{15.21}$$

where

$$\kappa \leq \left\lfloor 65[(WR+3D)^2] \right\rfloor,$$

$$D = \sqrt{\sum_{i=1}^{m}[\max\{0, \xi_i - \lambda\alpha_i k(\mathbf{x}_i,\mathbf{x}_i)\}]^2},$$

$$W^2 = \sum_{i,j=1}^{m} \alpha_i\alpha_j y_i y_j k(\mathbf{x}_i,\mathbf{x}_j),$$

provided $m \geq \max\{2/\epsilon, 6\}$ and $\kappa \leq em$.

Proof Firstly, AM–SVMs are linear classifiers $f(\mathbf{x}) = (\mathbf{w} \cdot \Phi(\mathbf{x}))$ where $\mathbf{w}$ is defined by Equation (15.16). We wish to redefine the measure of margin error $d((\mathbf{x}, y), \mathbf{w}, \rho) = \rho - y_i f(\mathbf{x}_i)$ in Theorem 15.2 in terms of ξ_i and $\lambda\alpha_i k(\mathbf{x}_i, \mathbf{x}_i)$ to capture the adaptive margin of a training point $\mathbf{x}_i$. Then we know from the assumption of a feasible solution $\boldsymbol{\alpha}, \boldsymbol{\xi}$ that

$$\max\{0, \rho - y_i f(\mathbf{x}_i))\} \leq \max\{0, \rho - 1 + \xi_i - \lambda\alpha_i k(\mathbf{x}_i, \mathbf{x}_i)\}\,. \tag{15.22}$$

In order to apply Theorem 15.2 for *any* vector $\mathbf{w}$ we have to normalize ρ, D, and $\boldsymbol{\alpha}$ by the norm of $\|\mathbf{w}\|_{\mathcal{F}} = W$ given by (15.16). This results in

$$\kappa = \left\lfloor \frac{65[(R + \frac{1}{W}D)^2 + 2.25\frac{1}{W}RD]}{\rho^2} W^2 \right\rfloor\,. \tag{15.23}$$

Now we fix $\rho = 1$ as done by AM–SVMs. This gives for Equation (15.22)

$$\max\{0, \rho - y_i f(\mathbf{x}_i))\} \leq \max\{0, \xi_i - \lambda\alpha_i k(\mathbf{x}_i, \mathbf{x}_i)\}\,. \tag{15.24}$$

Making use of

$$\left[\left(R + \frac{1}{W}D\right)^2 + 2.25\frac{1}{W}RD\right] W^2 \leq [(WR + 3D)^2]\,, \tag{15.25}$$

the theorem is proven. ■

From the theorem, one can gain the following insights. Our goal to minimize the generalization error is achieved by minimizing κ, the minimum of which is a tradeoff between minimizing W (the margin) and D (the loss with adaptive margin). We require a small value of both but small values of one term automatically gives a large value of the other. By minimizing $\sum_{i=1}^{m} \xi_i$ AM–SVMs effectively control the tradeoff between the two terms through the parameter λ. For small values of λ, the resulting D is small and W can take any value as it is not minimized (it can be forced to very large values). For large λ the increased margin in D acts a regularizer, penalizing large values of α. This results in small values of W (a smooth function) but large values of D (large training error). This bound motivates the objective function of AM–SVMs which at first appears to only minimize error and have no regularization. In fact, as we have seen, the regularization comes from the adaptive margin in the constraints controlled by λ.

15.6 Experiments

15.6.1 Artificial Data

15.6.1.1 LOO–SVMs

We first describe some two dimensional examples to illustrate how the new technique works. Let us first consider AM–SVMs with regularization parameter $\lambda = 1$ (this corresponds to LOO–SVMs, see Section 15.2). Figures 15.2 and 15.3 show two artificially constructed training problems with various solutions. We fixed $k(\cdot,\cdot)$ to be a radial basis function (RBF) kernel

$$k(\mathbf{x},\mathbf{x}') = \exp\left(-\|\mathbf{x}-\mathbf{x}'\|^2/(2\,\sigma^2)\right), \tag{15.26}$$

and then found the solution to the problems with LOO–SVM, which has no other free parameters, and with SVMs, for which one controls the *soft margin* with the free parameter $C = \frac{1}{\lambda}$. The first solution (left) for both training problems is the LOO–SVM solution and the other two solutions for each problem are SVMs with different choices of *soft margin* using parameter $C = 1$ (middle) and $C = 100$ (right).

In the first problem (Figure 15.2) the two classes (represented by crosses and dots) are almost linearly separable apart from a single outlier. The automatic *soft margin* control of LOO–SVMs constructs a classifier which incorrectly classifies the far right dot, assuming that it is an outlier. The Support Vector solutions both classify the outlier correctly resulting in non–smooth decision rules. In the second problem (Figure 15.3) the two classes occupy opposite sides (horizontally) of the picture, but slightly overlap. In this case the data is only separable with a highly nonlinear decision rule, as reflected in the solution by an SVM with parameter $C = 100$ (right). Both problems highlight the difficulty of choosing the parameter C in SVMs, whereas LOO–SVMs (AM–SVMs with $\lambda = 1$) appear to produce robust[6], natural decision rules.

15.6.1.2 AM–SVMs

In order to demonstrate how the regularization parameter λ in AM–SVMs (rather than being fixed to $\lambda = 1$ as in LOO-SVMs) affects the generated decision rule we give a comparison on the same toy problem as SVMs and LP-SVMs. We generated another two class problem in $\mathbb{R}^2$ (represented by crosses and dots) and trained an AM–SVM using RBF–kernels ($\sigma = 0.5$) with $\lambda = 1, 2, 5, 10$ (see Figure 15.4). As can be seen increasing λ allows AM–SVMs to widen the margin for points far away

6. As there is no unique definition of robustness (see, e.g., [Huber, 1981]) we call a classification learning algorithm *robust* if a few patterns far apart from the remaining ones (in the metric induced by Φ) have no influence on the resulting decision function.

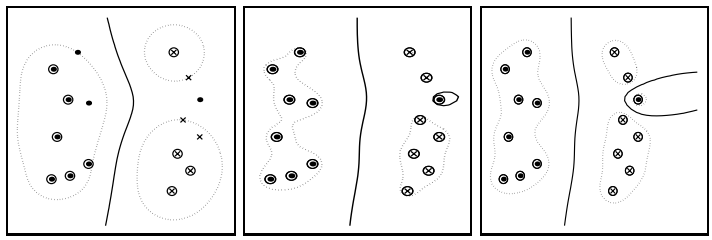


Figure 15.2 A simple two dimensional problem with one outlier solved by LOO-SVMs (left) and SVMs with $C = 1$ (middle) and $C = 100$ (right). LOO-SVMs soft margin regularization appears to perform better than the choices of parameter for SVMs.

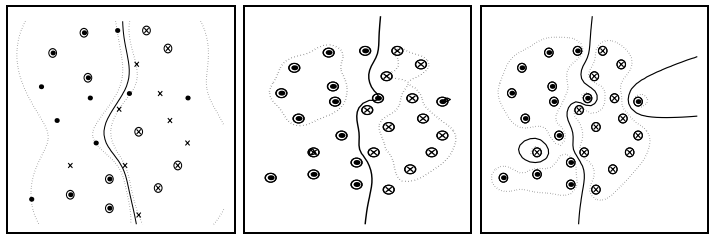


Figure 15.3 A simple two dimensional problem of two overlapping classes solved by LOO-SVMs (left) and SVMs with $C = 1$ (middle) and $C = 100$ (right). LOO-SVMs soft margin regularization appears to perform better than the choices of parameter for SVMs.

from the decision surface. Consequently, the algorithm is more robust to outliers which results in very smooth decision functions. In Figure 15.5 we used the same dataset and trained ν LP–SVMs [Graepel et al., 1999]. ν LP–SVMs are obtained by reparameterizing Equation (15.18) where ν upper–bounds the number of margin errors. Varying $\nu = 0.0, 0.1, 0.2, 0.5$ shows that *margin* errors are sacrificed in order to lower the complexity of the decision function f measured in the one–norm (see Equation (15.17) where λ can be replaced by a fixed function of ν). As already mentioned this leads to non–smooth functions. Furthermore it should be noted that the outlier (dot) on the far left side leads to very rugged decision functions. Similar conclusions can be drawn for ν SVMs [Schölkopf et al., 1998c] (see Figure 15.6) though the decision functions are smoother. Thus, AM–SVMs turn out to provide robust solutions (through control of the regularization parameter) which provide a new approach when compared to the solutions of SVMs and LP–SVMs. In these toy examples AM–SVMS appear to provide decision functions which are less influenced by single points (outliers).

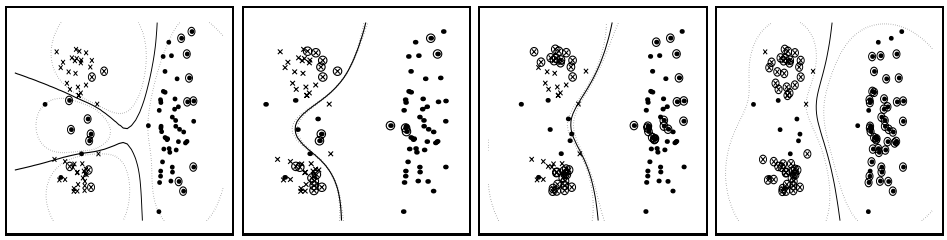


Figure 15.4 Decision functions (solid lines) obtained by AM–SVMs with different choices of the regularization parameter λ. The dashed line represents the minimal margin over all training points. **(a)** $\lambda = 1$ is equivalent to LOO–SVMs **(b)** $\lambda = 2$, **(c)** $\lambda = 5$, and **(d)** $\lambda = 10$ widens the amount to which margin errors at each point are accepted and thus results in very flat functions. Note, that less attention is paid to the outlier (dot) at the left hand side.

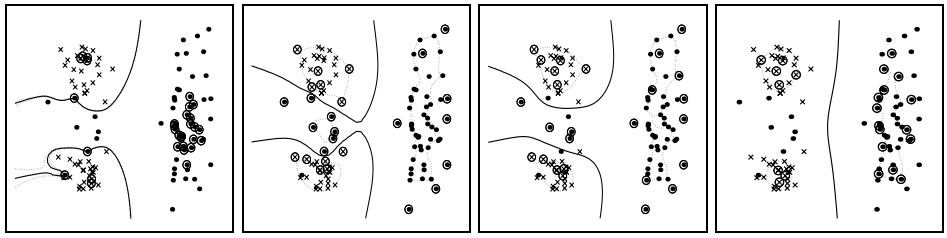


Figure 15.5 Decision functions (solid lines) obtained by ν LP–SVMs with different choices of the assumed noise level ν. The dashed line represents the margin. **(a)** $\nu = 0.0$ leads to very non–smooth and overfitted decision functions. **(b)** $\nu = 0.1$, **(c)** $\nu = 0.2$, and **(d)** $\nu = 0.5$ smooth the decision function.

15.6.2 Benchmark Datasets

We conducted computer simulations using 6 artificial and real world datasets from the UCI benchmark repository, following the same experimental setup as by Rätsch et al. [1998]. The authors of this article also provide a website to obtain the data.[7] Briefly, the setup is as follows: the performance of a classifier is measured by its average error over one hundred partitions of the datasets into training and testing sets. Free parameter(s) in the learning algorithm are chosen as the median value of the best model chosen by cross validation over the first five training datasets.

Table 15.1 compares percentage test error of LOO–SVMs to AdaBoost (AB), Regularized AdaBoost (AB_R) and SVMs which are all known to be excellent classifiers.[8] The competitiveness of LOO–SVMs to SVMs and AB_R (which both

7. http://svm.first.gmd.de/~raetsch/data/benchmarks.htm. The datasets have been preprocessed to have mean zero and standard deviation one, and the exact one hundred splits of training and testing sets used in the author's experiments can be obtained.
8. The results for AB, AB_R and SVMs were taken from [Rätsch et al., 1998]

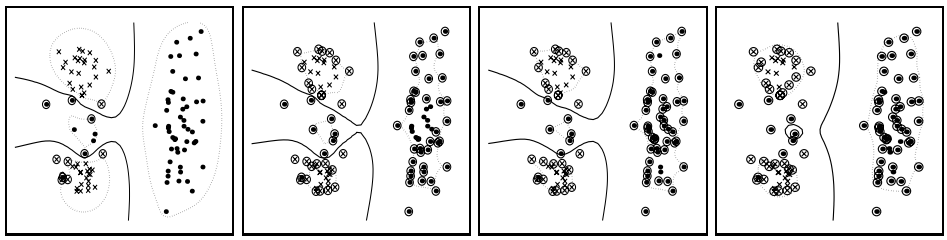


Figure 15.6 Decision functions (solid lines) obtained by ν SVMs with different choices of the assumed noise level ν. The dashed line represents the margin. **(a)** $\nu = 0.0$ leads to an overfitted decision functions (note the captured outlier in the lower left region). **(b)** $\nu = 0.1$, **(c)** $\nu = 0.2$, and **(d)** $\nu = 0.5$ allow for much flatter functions though regularizing differently to AM–SVMs.

	AB	AB_R	SVM	LOO–SVM
Banana	12.3	10.9	11.5	10.6
B. Cancer	30.4	26.5	26.0	26.3
Diabetes	26.5	23.9	23.5	23.4
Heart	20.3	16.6	16.0	16.1
Thyroid	4.4	4.4	4.8	5.0
Titanic	22.6	22.6	22.4	22.7

Table 15.1 Comparison of percentage test error of AdaBoost (AB), Regularized AdaBoost (AB_R), Support Vector Machines (SVMs) and Leave–One–Out SVMs (LOO–SVMs) on 6 datasets.

have a *soft margin* control parameter) is remarkable considering LOO–SVMs have no free parameter. This indicates that the *soft margin* automatically selected by LOO–SVMs is close to optimal. AdaBoost loses out to the three other algorithms, being essentially an algorithm designed to deal with noise-free data.

To give more insight into the behaviour of the algorithm we give two plots in Figure 15.7. The left graph shows the fraction of training points that have non–zero coefficients (*SVs*) plotted against $\log(\sigma)$ (*RBF width*) on the thyroid dataset. Here, one can see the sparsity of the decision rule, the sparseness of which depends on the chosen value of σ. The right graph shows the percentage training and test error (*train err* and *test err*), the value of $\sum_{i=1}^{m} \xi_i$ (*slacks*) and the value of the bound given in Theorem 15.1 (*l-o-o bound*). One can see the training and test error (and the bound) closely match. The minimum of all four plots is roughly at $\log(\sigma) = -1$, indicating one could perform model selection using one of the known expressions. Note also that for a reasonable range of σ the test error is roughly the same, indicating the *soft margin* control overcomes overfitting problems.

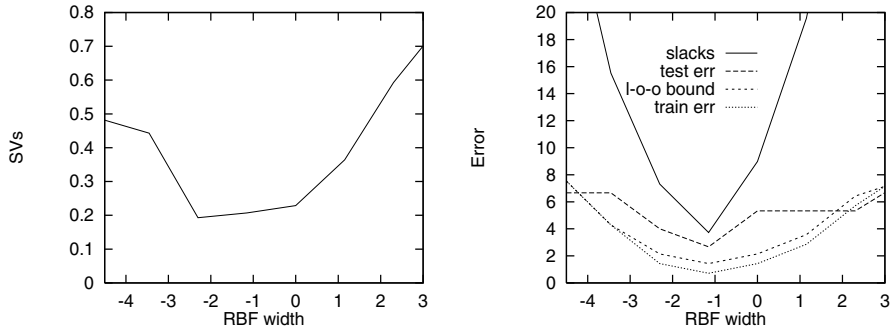


Figure 15.7 The fraction of training patterns that are Support Vectors (top) and various error rates (bottom) both plotted against RBF kernel width for Leave–One–Out Machines on the thyroid dataset.

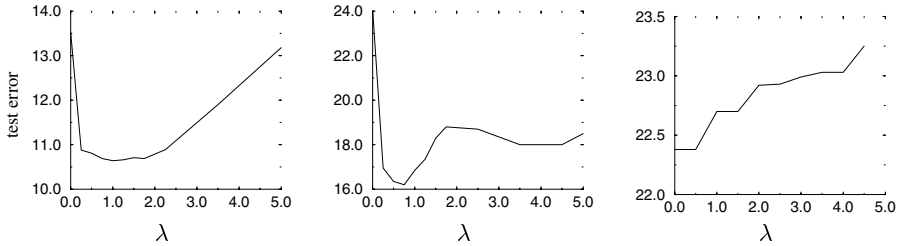


Figure 15.8 Test error plotted against the regularization parameter λ in AM–SVMs. The three plots from left to right are (a) the banana dataset, (b) heart dataset and (c) titanic dataset. Note how $\lambda = 1$ is close to the optimum of the bowl in the first two plots, but in the third plot the plot is not a bowl at all – the best choice of regularization is to choose no regularization ($\lambda = 0$).

Finally, we conducted experiments to assess the effect in generalization performance by controlling the regularization parameter λ in AM-SVMs. Figure 15.8 plots λ against test error for three of the datasets averaged over 10 runs for the first two, and over all 100 runs for the last. The banana dataset (left) and the heart dataset (middle) gave bowl-shaped graphs with the minimum exactly (banana) or almost (heart) at $\lambda = 1$. The optimum choice of λ for the titanic dataset, on the other hand, is at $\lambda = 0$. In this case the best choice of the regularization parameter λ is to have no regularization at all – the training points give enough information about the unknown decision function. Note this error rate for $\lambda = 0$ is as good as the best SVM solution (see Table 15.1). The first two plots and the results in Table 15.1 justify the choice of $\lambda = 1$ in LOO-SVMs. The last plot in Figure 15.8 justifies AM-SVMs.

15.7 Discussion

In this chapter we presented a new learning algorithm for kernel classifiers. Motivated by minimizing a bound on leave–one–out error we obtained LOO–SVMs and generalizing this approach to control regularization through the margin loss we obtained AM-SVMs. This approach introduced a novel method of capacity control via margin maximization by allowing adaptive rather than fixed margins at each training pattern. We have shown experimentally that this reformulation results in an algorithm which is robust against outliers. Nevertheless, our algorithm has a parameter λ which needs to be optimized for a given learning problem. Further investigations can be made in the derivation of bounds on the leave–one–out error of this algorithm which allows for efficient model order selection. Finally, we note that penalization of the diagonal of the kernel matrix is a well known technique in regression estimation known as Ridge Regression [Hoerl and Kennard, 1970].

Acknowledgments

The authors would like to thank Alex Gammerman, Thore Graepel, Tom Melluish, and Craig Saunders for discussions. In particular, we are indebted to both John Shawe-Taylor and Vladimir Vapnik for their help with this work. Ralf Herbrich would like to thank the Department of Computer Science at Royal Holloway for the warm hospitality during his research stay. Jason Weston thanks the ESPRC for providing financial support through grant GR/L35812.

16 GACV for Support Vector Machines

Grace Wahba
Department of Statistics
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706, USA
wahba@stat.wisc.edu

Yi Lin
Department of Statistics
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706, USA
yilin@stat.wisc.edu

Hao Zhang
Department of Statistics
University of Wisconsin
1210 West Dayton Street
Madison, WI 53706, USA
hzhang@stat.wisc.edu

We introduce the Generalized Approximate Cross Validation (GACV) for estimating tuning parameter(s) in SVMs. The GACV has as its target the choice of parameters which will minimize the Generalized Comparative Kullback-Leibler Distance (GCKL). The GCKL is seen to be an upper bound on the expected misclassification rate. Some modest simulation examples suggest how it might work in practice. The GACV is the sum of a term which is the observed (sample) GCKL plus a margin-like quantity.

16.1 Introduction

It is now common knowledge that the support vector machine (SVM) paradigm, which has proved highly successful in a number of classification studies, can be cast as a variational/regularization problem in a reproducing kernel Hilbert space (RKHS), see [Kimeldorf and Wahba, 1971, Wahba, 1990, Girosi, 1998, Poggio and Girosi, 1998], the papers and references in [Schölkopf et al., 1999a], and elsewhere. In this note, which is a sequel to [Wahba, 1999b], we look at the SVM paradigm from the point of view of a regularization problem, which allows a comparison with penalized log likelihood methods, as well as the application of model selection and tuning approaches which have been used with those and other regularization-type algorithms to choose tuning parameters in nonparametric statistical models.

reproducing kernel Hilbert space (RKHS)

We first note the connection between the SVM paradigm in RKHS and the (dual) mathematical programming problem traditional in SVM classification problems. We then review the Generalized Comparative Kullback-Leibler distance (GCKL) for the usual SVM paradigm, and observe that it is trivially a simple upper bound on the expected misclassification rate. Next we revisit the GACV (Generalized Approximate Cross Validation) as a proxy for the GCKL proposed by Wahba [1999b] and the argument that it is a reasonable estimate of the GCKL. We found that it is not necessary to do the randomization of the GACV in [Wahba, 1999b], because it can be replaced by an equally justifiable approximation which is readily computed exactly, along with the SVM solution to the dual mathematical programming problem. This estimate turns out interestingly, but not surprisingly to be simply related to what several authors have identified as the (observed) VC dimension of the estimated SVM. Some preliminary simulations are suggestive of the fact that the minimizer of the GACV is in fact a reasonable estimate of the minimizer of the GCKL, although further simulation and theoretical studies are warranted. It is hoped that this preliminary work will lead to better understanding of "tuning" issues in the optimization of SVM's and related classifiers.

GCKL

GACV

16.2 The SVM Variational Problem

Let $\mathcal{T}$ be an index set, $t \in \mathcal{T}$. Usually $\mathcal{T} = E^d$, Euclidean d-space, but not necessarily. Let $K(s,t), s,t \in \mathcal{T}$, be a positive definite function on $\mathcal{T} \otimes \mathcal{T}$, and let $\mathcal{H}_K$ be the RKHS with reproducing kernel (RK) K. See [Wahba, 1990, 1982, Lin et al., 1998] for more on RKHS. RK's which are tensor sums and products of RK's are discussed there and elsewhere. K may contain one or more tuning parameters, to be chosen. A variety of RK's with success in practical applications have been proposed by various authors, see, e.g., the Publications list at `http://www.kernel-machines.org`. Recently [Poggio and Girosi, 1998] interestingly observed how different scales may be accommodated using RKHS methods. We are given a training set $\{y_i, t_i\}$, where the attribute vector $t_i \in \mathcal{T}$, and $y_i = \pm 1$ according as an example with attribute

reproducing kernel

regularization problem

vector t_i is in category $\mathcal{A}$ or $\mathcal{B}$. The classical SVM paradigm is equivalent to: find f_λ of the form $const + h$, where $h \in \mathcal{H}_K$ to minimize

$$\frac{1}{n}\sum_{i=1}^{n}(1 - y_i f_i)_+ + \lambda \|h\|^2_{\mathcal{H}_K}, \tag{16.1}$$

here $f_i = f(t_i)$, and $(\tau)_+ = \tau, \tau > 0; = 0$ otherwise. Similar regularization problems have a long history, see, for example [Kimeldorf and Wahba, 1971]. Once the minimizer, call it f_λ is found, then the decision rule for a new example with attribute vector t is: $\mathcal{A}$ if $f_\lambda(t) > 0$, $\mathcal{B}$ if $f_\lambda(t) < 0$.

We will assume for simplicity that K is strictly positive definite on $\mathcal{T} \otimes \mathcal{T}$, although this is not necessary. The minimizer of (16.1) is known to be in the span $\{K(\cdot, t_i), i = 1, \cdots n\}$, of representers of evaluation in $\mathcal{H}_K$. The function $K(\cdot, t_i)$ is $K(s, t_i)$ considered as a function of s with t_i fixed. The famous "reproducing" property gives the inner product in $\mathcal{H}_K$ of two representers as $< K(\cdot, t_i), K(\cdot, t_j) >_{\mathcal{H}_K} = K(t_i, t_j)$. Thus, if $h(\cdot) = \sum_{i=1}^{n} c_i K(\cdot, t_i)$, then $\|h\|^2_{\mathcal{H}_K} = \sum_{i,j=1}^{n} c_i c_j K(t_i, t_j)$. Letting $e = (1, \cdots, 1)', y = (y_1, \cdots, y_n)', c = (c_1, \cdots, c_n)', (f(t_1), \cdots f(t_n))' = (f_1, \cdots, f_n)'$, and with some abuse of notation, letting $f = (f_1, \cdots, f_n)'$ and K now be the $n \times n$ matrix with ijth entry $K(t_i, t_j)$, and noting that $f(t) = d + \sum_{i=1}^{n} c_i K(t, t_i)$ for some c, d, we have

$$f = Kc + ed \tag{16.2}$$

and the variational problem (16.1) becomes: find (c, d) to minimize

$$\frac{1}{n}\sum_{i=1}^{n}(1 - y_i f_i)_+ + \lambda c' K c. \tag{16.3}$$

16.3 The Dual Problem

Let Y be the $n \times n$ diagonal matrix with y_i in the iith position, and let $H = \frac{1}{2n\lambda} YKY$. By going to the dual form of (16.3), it can be shown that $c = \frac{1}{2n\lambda} Y\alpha$, where α is the solution to the problem

$$\text{maximize } L = -\frac{1}{2}\alpha' H \alpha + e'\alpha \tag{16.4}$$

$$\text{subject to } \begin{cases} 0 & \leq & \alpha & \leq & e \\ e'Y\alpha & = & y'\alpha & = & 0. \end{cases} \tag{16.5}$$

Assuming that there is an i for which $0 < \alpha_i < 1$, it can also be shown that $d = 1/y_i - \sum_{j=1}^{n} c_j K(t_i, t_j)$. This is the usual form in which the SVM is computed. In the experiments reported below, we used the MINOS [Murtagh and Saunders, 1998] optimization routine to find α, and hence c. The support vectors are those $K(\cdot, t_i)$ for which $\alpha_i \neq 0$, equivalently $c_i \neq 0$. d can be found from any of the support vectors for which $0 < \alpha_i < 1$.

For future reference we review the relation between the (hard) margin (γ) of the support vector machine classifier and $\sum_{y_i f_{\lambda i} \le 1} \alpha_{\lambda i}$. In the situation where we can separate the training set points perfectly, γ is given by

$$\gamma^2 = 2n\lambda \left(\sum_{y_i f_{\lambda i} \le 1} \alpha_{\lambda i} \right)^{-1} . \tag{16.6}$$

See [Cortes and Vapnik, 1995, Bartlett and Shawe-Taylor, 1999]. (Notice the notation is a bit different from ours in these papers.) By definition the margin of the (hard margin) support vector machine classifier is $\gamma = \frac{1}{\|h\|_{\mathcal{H}_K}} = (c'Kc)^{-1/2}$. The equality (16.6) can be seen from the following: In the perfectly separable case, where all members of the training set are classified correctly, $\alpha_{\lambda i}$ is the solution of the problem below:

margin of the SVM classifier

$$\text{maximize} \quad L = -\frac{1}{2}\alpha' H \alpha + e'\alpha \tag{16.7}$$

$$\text{subject to} \quad \alpha_i \ge 0 \text{ and } y'\alpha = 0. \tag{16.8}$$

Introducing the Lagrangian multipliers $\xi = (\xi_1, \ldots, \xi_n)'$ and β for the constraints, the Lagrangian for this problem is

$$L_P = -\frac{1}{2}\alpha' H \alpha + e'\alpha - \beta y'\alpha - \xi'\alpha$$

and $\alpha_{\lambda i}$ satisfies the Kuhn-Tucker conditions:

$$\begin{aligned} \frac{\partial}{\partial \alpha} L_P = -H\alpha + e - \beta y - \xi &= 0 \\ \alpha_i &\ge 0, \ i = 1, 2, ..., n \\ y'\alpha &= 0 \\ \xi_i &\ge 0, \ i = 1, 2, ..., n \\ \xi_i \alpha_i &= 0, \ i = 1, 2, ..., n \end{aligned}$$

From these and the relation that $c = Y\alpha_\lambda/(2n\lambda)$, it is easy to get

$$c'Kc = \frac{1}{2n\lambda}\alpha'_\lambda H \alpha_\lambda = \frac{1}{2n\lambda}\left[\alpha'_\lambda e - \beta \alpha'_\lambda y - \alpha'_\lambda \xi\right] = \frac{1}{2n\lambda}\left[\alpha'_\lambda e\right]. \tag{16.9}$$

Since $\alpha_{\lambda i} = 0$ if $y_i f_i > 1$, we finally get

$$\gamma^2 = (c'Kc)^{-1} = 2n\lambda \left[\sum_{y_i f_{\lambda i} \le 1} \alpha_{\lambda i} \right]^{-1} .$$

16.4 The Generalized Comparative Kullback-Leibler Distance

Suppose unobserved y_i's will be generated according to an (unknown) probability model with $p(t) = p_{\text{true}}(t)$ being the probability that an instance with attribute vector t is in class $\mathcal{A}$. Let y_j be an (unobserved) value of y associated with t_j.

GCKL

Given f_λ, define the Generalized Comparative Kullback-Leibler distance (GCKL distance) with respect to g as

$$GCKL(p_{\text{true}}, f_\lambda) \doteq GCKL(\lambda) = E_{\text{true}} \frac{1}{n} \sum_{j=1}^{n} g(y_j f_{\lambda j}). \tag{16.10}$$

penalized log likelihood

Here f_λ is considered fixed and the expectation is taken over future, unobserved y_j. If $g(\tau) = \ln(1 + e^{-\tau})$, (which corresponds to classical penalized log likelihood estimation if it replaces $(1-\tau)_+$ in (16.1)) $GCKL(\lambda)$ reduces to the usual CKL for Bernoulli data[1] averaged over the attribute vectors of the training set. More details may be found in [Wahba, 1999b]. Let $[\tau]_* = 1$ if $\tau > 0$ and 0 otherwise. If $g(\tau) = [-\tau]_*$, then

$$E_{\text{true}}[-y_j f_{\lambda j}]_* = p_{[\text{true}]j}[-f_{\lambda j}]_* + (1 - p_{[\text{true}]j})[f_{\lambda j}]_* \tag{16.11}$$

$$= p_{[\text{true}]j}, \quad f_{\lambda j} < 0 \tag{16.12}$$

$$= (1 - p_{[\text{true}]j}), \quad f_{\lambda j} > 0, \tag{16.13}$$

where $p_{[\text{true}]j} = p_{[true]}(t_j)$, so that the $GCKL(\lambda)$ is the expected misclassification rate for f_λ on unobserved instances if they have the same distribution of t_j as the training set. Similarly, if $g(\tau) = (1-\tau)_+$, then

$$E_{\text{true}}(1 - y_j f_{\lambda j})_+ = p_{[\text{true}]j}(1 - f_{\lambda j}), \quad f_{\lambda j} < -1 \tag{16.14}$$

$$= 1 + (1 - 2p_{[\text{true}]j}) f_{\lambda j}, \quad -1 \le f_{\lambda j} \le 1 \tag{16.15}$$

$$= (1 - p_{[\text{true}]j})(1 + f_{\lambda j}), \quad f_{\lambda j} > 1. \tag{16.16}$$

Note that $[-y_i f_i]_* \le (1 - y_i f_i)_+$, so that the $GCKL$ for $(1 - y_i f_i)_+$ is an *upper bound* for the *expected misclassification rate* - see Figure 16.1.

16.5 Leaving-out-one and the GACV

Recently there has been much interest in choosing λ (or its equivalent, referred to in the literature as $\frac{1}{2nC}$), as well as other parameters inside K. See for example [Burges, 1998, Cristianini et al., 1999, Kearns et al., 1997], surely not a complete list. Important references in the statistics literature that are related include [Efron and Tibshirani, 1997, Ye and Wong, 1997]. Lin et al. [1998] consider in detail the case $g(\tau) = \ln(1 + e^{-\tau})$. We now obtain the GACV estimate for λ and other tuning parameters.

1. The usual CKL (comparative Kullback-Leibler distance) is the Kullback-Leibler distance plus a term which depends only on $p_{[true]}$. In this case g is the negative log likelihood and f_λ plays the role of (an estimate of) the logit $ln[p/1-p]$. See also [Friedman et al., 1998].

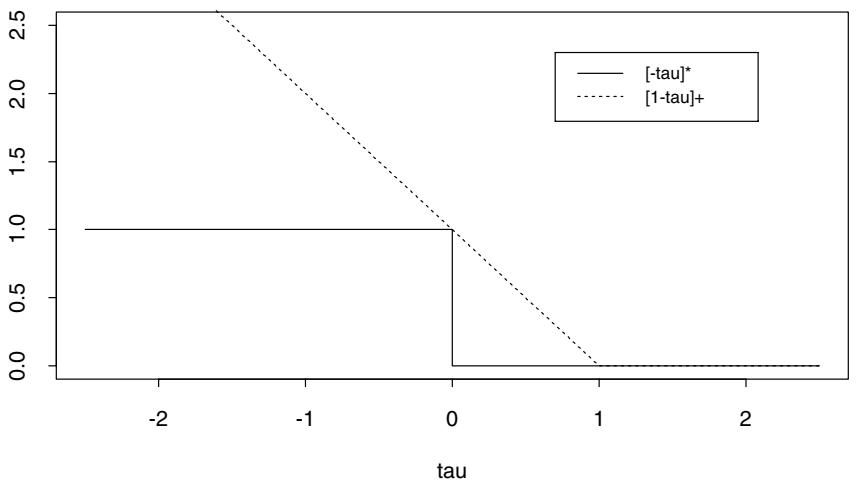


Figure 16.1 $g(\tau) = (1-\tau)_+$ and $g(\tau) = [-\tau]_*$ compared.

Let $f_\lambda^{[-i]}$ be the solution to the variational problem: find f of the form $f = const + h$ with $h \in \mathcal{H}_K$ to minimize

$$\frac{1}{n}\sum_{\substack{j=1\\ j\neq i}}^{n} g(y_j f_j) + \lambda \|h\|^2_{\mathcal{H}_K}. \tag{16.17}$$

leaving-out-one

Then the leaving-out-one function $V_0(\lambda)$ is defined as

$$V_0(\lambda) = \frac{1}{n}\sum_{i=1}^{n} g(y_i f_{\lambda i}^{[-i]}). \tag{16.18}$$

Since $f_{\lambda i}^{[-i]}$ does not depend on y_i but is (presumably) on average close to $f_{\lambda i}$, we may consider $V_0(\lambda)$ a proxy for $GCKL(\lambda)$, albeit one that is not generally feasible to compute in large data sets. Now let

$$V_0(\lambda) = OBS(\lambda) + D(\lambda), \tag{16.19}$$

where $OBS(\lambda)$ is the observed match of f_λ to the data,

$$OBS(\lambda) = \frac{1}{n}\sum_{i=1}^{n} g(y_i f_{\lambda i}) \tag{16.20}$$

and

$$D(\lambda) = \frac{1}{n}\sum_{i=1}^{n} [g(y_i f_{\lambda i}^{[-i]}) - g(y_i f_{\lambda i})]. \tag{16.21}$$

Using a first order Taylor series expansion gives

$$D(\lambda) \approx -\frac{1}{n}\sum_{i=1}^{n}\frac{\partial g}{\partial f_{\lambda i}}(f_{\lambda i} - f_{\lambda i}^{[-i]}). \tag{16.22}$$

Next we let $\mu(f)$ be a "prediction" of y given f. Here we let

$$\mu_i = \mu(f_i) = \sum_{y\in\{+1,-1\}} \frac{\partial}{\partial f_i} g(y_i f_i). \tag{16.23}$$

When $g(\tau) = \ln(1+e^{-\tau})$ then $\mu(f) = 2p-1 = E\{y|p\}$. Since this $g(\tau)$ corresponds to the penalized log likelihood estimate, it is natural in this case to define the "prediction" of y given f as the expected value of y given f (equivalently, p). For $g(\tau) = (1-\tau)_+$, this definition results in $\mu(f) = -1, f < -1; \mu(f) = 0, -1 \le f \le 1$ and $\mu(f) = 1$ for $f > 1$. This might be considered a kind of all-or-nothing prediction of y, being, essentially, ± 1 outside of the margin and 0 inside it. Letting $\mu_{\lambda i} = \mu(f_{\lambda i})$ and $\mu_{\lambda i}^{[-i]} = \mu(f_{\lambda i}^{[-i]})$, we may write (ignoring, for the moment, the possibility of dividing by 0),

$$D(\lambda) \approx -\frac{1}{n}\sum_{i=1}^{n}\frac{\partial g}{\partial f_{\lambda i}}\frac{(f_{\lambda i} - f_{\lambda i}^{[-i]})}{(y_i - \mu_{\lambda i}^{[-i]})}(y_i - \mu_{\lambda i}^{[-i]}) \tag{16.24}$$

This is equation (6.36) in [Wahba, 1999b]. We now provide somewhat different arguments than in [Wahba, 1999b] to obtain a similar result, which, however is easily computed as soon as the dual variational problem is solved.

Let $f_\lambda[i,x]$ be the solution of the variational problem (16.1) [2] given the data $\{y_1,\cdots,y_{i-1},x,y_{i+1},\cdots,y_n\}$. Note that the variational problem does not require that $x = \pm 1$. Thus $f_\lambda[i,y_i](t_i) \equiv f_{\lambda i}$. To simplify the notation, let $f_\lambda[i,x](t_i) = f_{\lambda i}[i,x] = f_{\lambda i}[x]$. In [Wahba, 1999b] it is shown, via a generalized leaving-out-one lemma, that $\mu(f)$ as we have defined it has the property that $f_{\lambda i}^{[-i]} = f_\lambda[i,\mu_{\lambda i}^{[-i]}](t_i)$. Letting $\mu_{\lambda i}^{[-i]} = x$, this justifies the approximation

$$\frac{f_{\lambda i} - f_{\lambda i}^{[-i]}}{y_i - \mu_{\lambda i}^{[-i]}} \equiv \frac{f_{\lambda i}[y_i] - f_{\lambda i}[x]}{y_i - x} \approx \frac{\partial f_{\lambda i}}{\partial y_i}. \tag{16.25}$$

Furthermore, $\mu_{\lambda i}^{[-i]} \equiv \mu(f_{\lambda i}^{[-i]}) = \mu(f_{\lambda i})$ whenever $f_{\lambda i}^{[-i]}$ and $f_{\lambda i}$ are both in the interval $(-\infty,-1)$, or $[-1,1]$, or $(1,\infty)$, which can be expected to happen with few exceptions. Thus, we make the further approximation $(y_i - \mu_{\lambda i}^{[-i]}) \approx (y_i - \mu_{\lambda i})$, and we replace (16.24) by

$$D(\lambda) \approx -\frac{1}{n}\sum_{i=1}^{n}\frac{\partial g}{\partial f_{\lambda i}}\frac{\partial f_{\lambda i}}{\partial y_i}(y_i - \mu_{\lambda i}). \tag{16.26}$$

---

2. d is not always uniquely determined; this however does not appear to be a problem in practice, and we shall ignore it.

Now, for $g(\tau) = (1-\tau)_+$

$$\begin{aligned}\frac{\partial g}{\partial f_{\lambda i}}(y_i - \mu_{\lambda i}) &= -2,\ y_i f_{\lambda i} < -1 \\ &= -1,\ y_i f_{\lambda i} \in [-1,1] \\ &= \ \ 0,\ y_i f_{\lambda i} > 1,\end{aligned}$$

giving finally

$$D(\lambda) \approx \frac{1}{n} \sum_{y_i f_{\lambda i} < -1} 2\frac{\partial f_{\lambda i}}{\partial y_i} + \frac{1}{n} \sum_{y_i f_{\lambda i} \in [-1,1]} \frac{\partial f_{\lambda i}}{\partial y_i}. \tag{16.27}$$

It is not hard to see how $\frac{\partial f_{\lambda i}}{\partial y_i}$ should be interpreted. Fixing λ and solving the variational problem for f_λ we obtain $\alpha = \alpha_\lambda$, $c = c_\lambda = \frac{1}{2n\lambda} Y\alpha_\lambda$ and for the moment letting f_λ be the column vector with ith component $f_{\lambda i}$, we have $f_\lambda = Kc_\lambda + ed = \frac{1}{2n\lambda} KY\alpha_\lambda + ed$. From this we may write

$$\frac{\partial f_{\lambda i}}{\partial y_i} = K(t_i, t_i)\frac{\alpha_{\lambda i}}{2n\lambda} \equiv \|K(\cdot, t_i)\|^2_{\mathcal{H}_K} \frac{\alpha_{\lambda i}}{2n\lambda}. \tag{16.28}$$

The resulting $GACV(\lambda)$, which is believed to be a reasonable proxy for $GCKL(\lambda)$, is, finally

GACV

$$GACV(\lambda) = \frac{1}{n}\sum_{i=1}^{n}(1 - y_i f_{\lambda i})_+ + \hat{D}(\lambda), \tag{16.29}$$

where

$$\hat{D}(\lambda) = \frac{1}{n}\left[2 \sum_{y_i f_{\lambda i} < -1} \frac{\alpha_{\lambda i}}{2n\lambda} \cdot \|K(\cdot, t_i)\|^2_{\mathcal{H}_K} + \sum_{y_i f_{\lambda i} \in [-1,1]} \frac{\alpha_{\lambda i}}{2n\lambda} \cdot \|K(\cdot, t_i)\|^2_{\mathcal{H}_K}\right]. \tag{16.30}$$

If $K = K_\theta$, where θ are some parameters inside K to which the result is sensitive, then we may let $GACV(\lambda) = GACV(\lambda, \theta)$. Note the relationship between $\hat{D}$ and $\sum_{y_i f_{\lambda i} \le 1} \alpha_{\lambda i}$ and the margin γ. If $K(\cdot,\cdot)$ is a radial basis function then $\|K(\cdot, t_i)\|^2_{\mathcal{H}_K} = K(0,0)$. Furthermore $\|K(\cdot, t_i) - K(\cdot, t_j)\|^2_{\mathcal{H}_K}$ is bounded above by $2K(0,0)$. If all members of the training set are classified correctly then $y_i f_i > 0$ and the sum following the 2 in (16.30) does not appear and $\hat{D}(\lambda) = K(0,0)/n\gamma^2$.

We note that Opper and Winther (Chapter 17) have obtained a different approximation for $f_{\lambda i} - f_{\lambda i}^{[-i]}$.

16.6 Numerical Results

We give two rather simple examples. For the first example, attribute vectors t were generated according to a uniform distribution on $\mathcal{T}$, the square depicted in Figure 16.2. The points outside the larger circle were randomly assigned $+1$ ("$+$") with probability $p_{[true]} = .95$ and -1 ("o") with probability .05. The points between the outer and inner circles were assigned $+1$ with probability $p_{[true]} = .50$, and the

points inside the inner circle were assigned +1 with probability $p_{[true]} = .05$. In this and the next example, $K(s,t) = e^{-\frac{1}{2\sigma^2}\|s-t\|^2}$, where σ is a tunable parameter to be chosen. Figure 16.3 gives a plot of $\log_{10}(GACV)$ of (16.29) and $\log_{10}(GCKL)$ of (16.10) as a function of $\log_{10}\lambda$, for $\log_{10}\sigma = -1$. Figure 16.4 gives the corresponding plot as a function of $\log_{10}\sigma$ for $\log_{10}\lambda = -2.5$, which was the minimizer of $\log_{10}(GACV)$ in Figure 16.3. Figure 16.5 shows the level curve for $f_\lambda = 0$ for $\log_{10}\lambda = -2.5$ and $\log_{10}\sigma = -1.0$, which was the minimizer of $\log_{10}(GACV)$ over the two plots. This can be compared to the theoretically optimal classifier, which the Neyman-Pearson Lemma says would be any curve between the inner and outer circles, where the theoretical log-odds ratio is 0. For the second example, Figure 16.6 corresponds to Figure 16.2, with $p_{[true]} = .95, .5$ and $.05$ respectively in the three regions, starting from the top. Figure 16.7 gives a plot of $\log_{10}(GACV)$ and $\log_{10}(GCKL)$ as a function of $\log_{10}\lambda$ for $\log_{10}\sigma = -1.25$ and Figure 16.8 gives $\log_{10}(GACV)$ and $\log_{10}(GCKL)$ as a function of $\log_{10}\sigma$ for $\log_{10}\lambda = -2.5$, which was the minimizer of Figure 16.7. Figure 16.9 gives the level curves for f_λ at 0 for $\log_{10}\lambda = -2.5$, $\log_{10}\sigma = -1.25$, which was the minimizer of $\log_{10}(GACV)$ over Figures 16.7 and 16.8. This can also be compared to the theoretically optimal classifier, which would be any curve falling between the two sine waves of Figure 16.7.

It can be seen that $\log_{10} GACV$ tracks $\log_{10} GCKL$ very well in Figures 16.3, 16.4, 16.7 and 16.8, more precisely, the minimizer of $\log_{10} GACV$ is a good estimate of the minimizer of $\log_{10} GCKL$.

A number of cross-sectional curves were plotted, first in $\log_{10}\lambda$ for a trial value of $\log_{10}\sigma$ and then in $\log_{10}\sigma$ for the minimizing value of $\log_{10}\lambda$ (in the GACV curve), and so forth, to get to the plots shown. A more serious effort to obtain the global minimizers over of $\log_{10}(GACV)$ over $\log_{10}\lambda$ and $\log_{10}\sigma$ is hard to do since both the $GACV$ and the $GCKL$ curves are quite rough. The curves have been obtained by evaluating the functions at increments on a log scale of .25 and joining the points by straight line segments. However, these curves (or surfaces) are not actually continuous, since they may have a jump (or tear) whenever the active constraint set changes. This is apparently a characteristic of generalized cross validation functions for constrained optimization problems when the solution is not a continuously differentiable function of the observations, see, for example [Wahba, 1982, Figure 7]. In practice, something reasonably close to the minimizer can be expected to be adequate.

Work is continuing on examining the $GACV$ and the $GCKL$ in more complex situations.

Acknowledgments

The authors thank Fangyu Gao and David Callan for important suggestions in this project. This work was partly supported by NSF under Grant DMS-9704758 and NIH under Grant R01 EY09946.

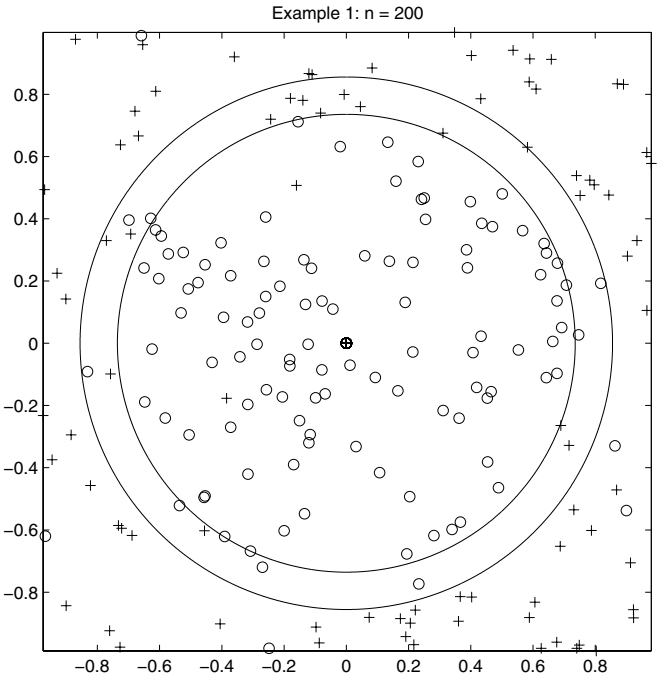


Figure 16.2 Data for Example 1, With Regions of Constant (Generating) Probability.

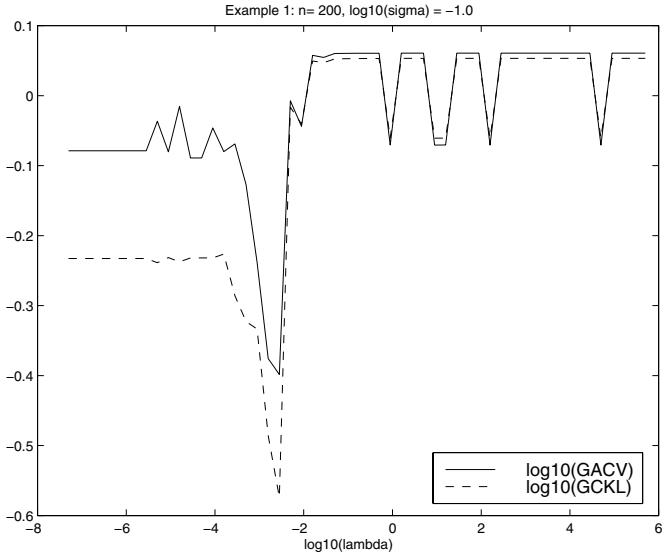


Figure 16.3 Plot of $\log_{10} GACV$ and $\log_{10} GCKL$ as a function of $\log_{10} \lambda$ for $\log_{10} \sigma = -1.0$.

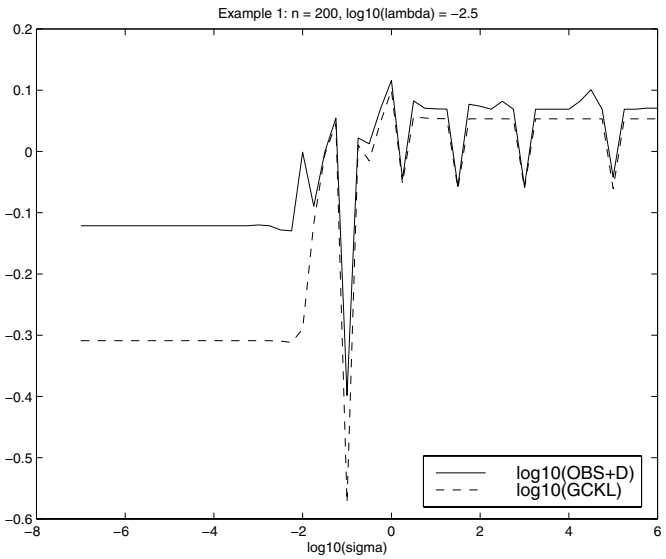


Figure 16.4 Plot of $\log_{10} GACV$ and $\log_{10} GCKL$ as a function of $\log_{10} \sigma$ for $\log_{10} \lambda = -2.5$.

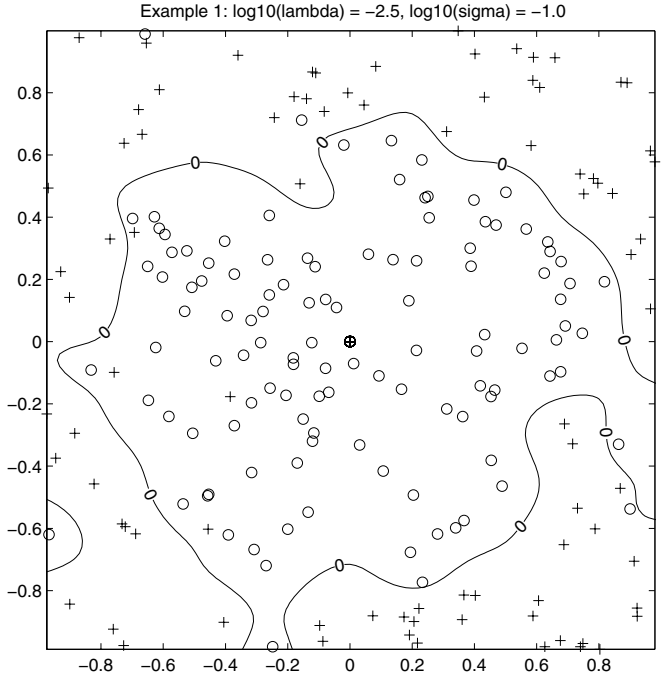


Figure 16.5 Level curve for $f_\lambda = 0$.

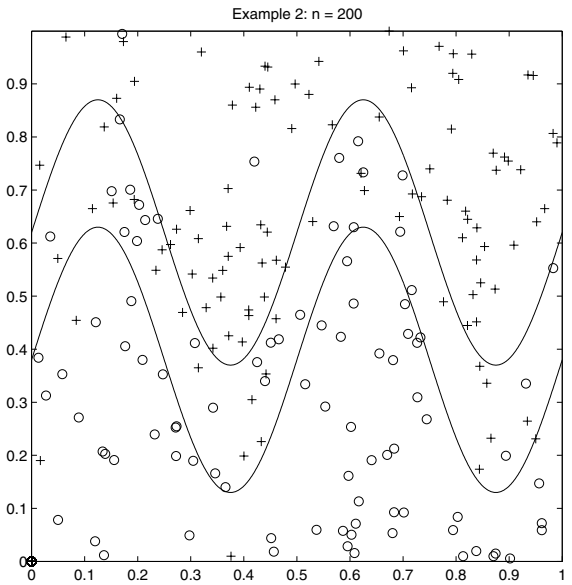


Figure 16.6 Data for Example 2, and Regions of Constant (Generating) Probability.

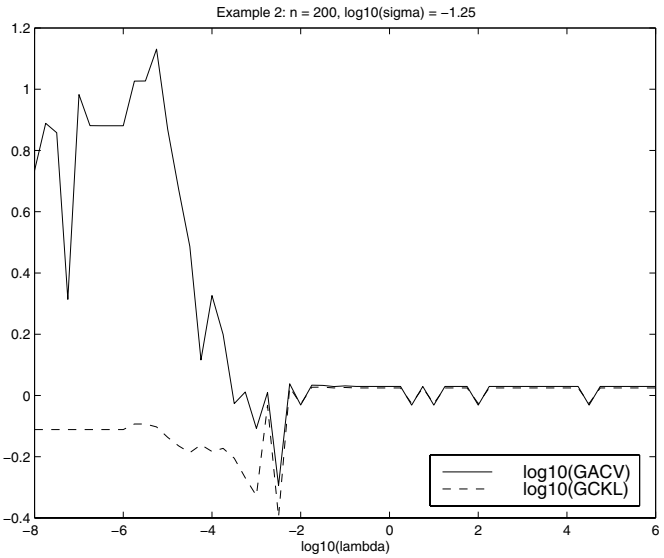


Figure 16.7 Plot of $\log_{10} GACV$ and $\log_{10} GCKL$ as a function of $\log_{10} \lambda$ for $\log_{10} \sigma = -1.25$.

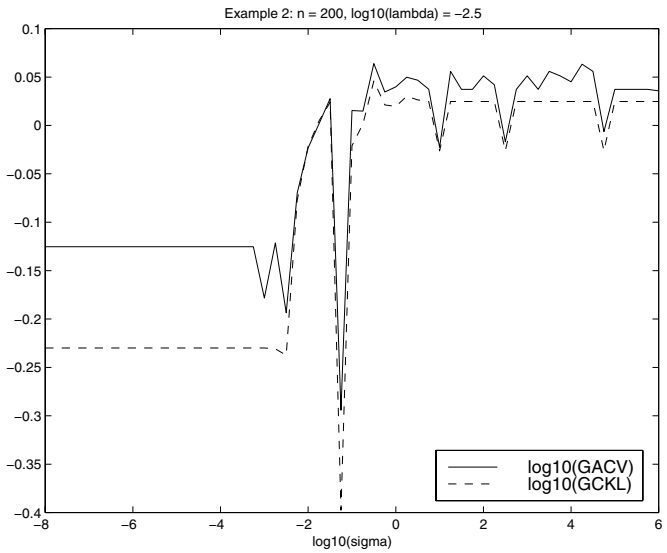


Figure 16.8 Plot of $\log_{10} GACV$ and $\log_{10} GCKL$ as a function of $\log_{10} \sigma$ for $\log_{10} \lambda = -2.5$.

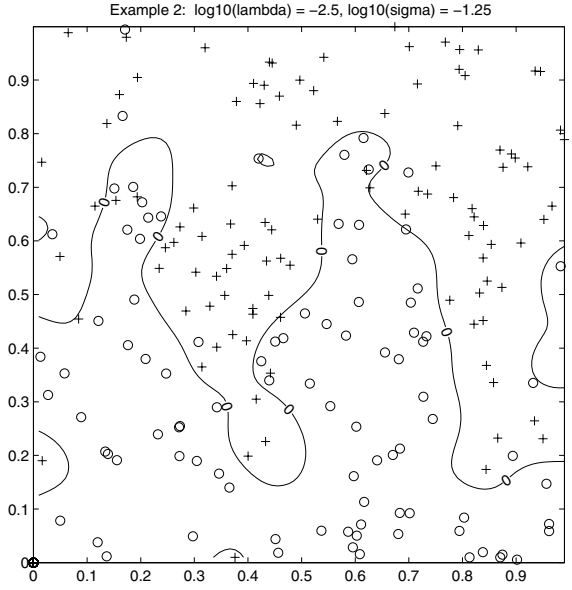


Figure 16.9 Level curve for $f_\lambda = 0$.

17 Gaussian Processes and SVM: Mean Field and Leave-One-Out

Manfred Opper
Department of Computer Science and Applied Mathematics,
Aston University
Birmingham B4 7ET
United Kingdom
m.opper@aston.ac.uk
http://neural-server.aston.ac.uk/People/opperm/

Ole Winther
Theoretical Physics II, Lund University
Sölvegatan 14 A
S-223 62 Lund
Sweden
winther@thep.lu.se
http://www.thep.lu.se/tf2/staff/winther/

In this chapter, we elaborate on the well-known relationship between Gaussian processes (GP) and Support Vector Machines (SVM). Secondly, we present approximate solutions for two computational problems arising in GP and SVM. The first one is the calculation of the posterior mean for GP classifiers using a "naive" mean field approach. The second one is a leave-one-out estimator for the generalization error of SVM based on a linear response method. Simulation results on a benchmark dataset show similar performances for the GP mean field algorithm and the SVM algorithm. The approximate leave-one-out estimator is found to be in very good agreement with the exact leave-one-out error.

17.1 Introduction

It is well-known that Gaussian Processes (GP) and Support Vector Machines (SVM) are closely related, see, e.g., [Wahba, 1999b, Williams, 1998]. Both approaches are non-parametric. This means that they allow (at least for certain kernels) for infinitely many parameters to be tuned, but increasing with the amount of data, only a finite number of them are active. Both types of models may be understood as generalizations of single layer perceptrons, where each input node to the perceptron computes a distinct nonlinear feature of the original inputs to the machine. In principle, the number of such features (and of the corresponding perceptron weights) can be arbitrarily large. However, by the specific training method, such vast increase in complexity does not necessarily result in overfitting.

For the support vector machine (in its simplest version), a quadratic optimization algorithm maximizes the gap between positive and negative examples. A simple mathematical analysis of this optimization problem shows that all the weights become just linear combinations of the input feature vectors. Hence, the corresponding coefficients in this combination are the new parameters to be calculated. Their number never exceeds the number of examples. Moreover, it is not necessary to evaluate the many nonlinear feature vectors during the calculations, but all calculations are expressed by the kernel function which is the inner product of two vectors of features at different input points. In fact, one need not even specify the non-linear features explicitly, but any positive semidefinite kernel function will implicitly define such features (see Chapter 1 for details).

A second way to regularize this problem comes from the Bayesian approach. Here, one introduces a prior distribution over the perceptron weights, which puts a smaller weight on the more complex features. If the prior distribution is a multivariate Gaussian (in the simplest case, just a product of univariate ones), the activation function of the single layer perceptron becomes a Gaussian process. Although a derivation of covariance functions based on a limiting process of multilayer networks is possible [Neal, 1996, Williams, 1997], one often simply uses a parameterized covariance function instead. Besides the simple fact that any kernel function used in the SVM approach can be used as a covariance function of the Gaussian process approach and vice versa, there are more striking mathematical relations between the two approaches as we will discuss in following.

This chapter deals with two subjects. First, we will show how SVM can be understood as the maximum a posteriori (MAP) prediction from GP using a certain non-normalized likelihood. The second part deals with two approximation techniques that are useful in performing calculations for SVM or GP which would otherwise be intractable or time consuming. We will discuss a linear response method to derive an approximate leave-one-out estimator for the generalization error of SVM. Mean field methods (which have been originally developed within statistical mechanics) can be used to cope with posterior averages for GP which are not analytically tractable.

The rest of the chapter is organized as follows. Section 17.2 reviews the Gaussian process approach to noise-free classification. In Section 17.3, we discuss how to extend this to modeling with noise. Section 17.4 deals with the relation of SVM to the maximum a posteriori prediction of GP. In Section 17.5, we derive a leave-one-out estimator for the generalization error using linear response theory and a (mean field) assumption. Section 17.6 reviews the "naive" mean field approach to Gaussian process classification. SVM and the naive mean field algorithm are compared in simulations in Section 17.7. The chapter is concluded in Section 17.8.

17.2 Gaussian Process Classification

Gaussian processes give a natural formulation of Bayesian learning in terms of prior distributions over functions. Here, we give a short summary of the basic concepts of Bayesian learning as applied to Gaussian Processes.

We consider a binary classifier with output $g(\mathbf{x}) = \mathrm{sgn} f(\mathbf{x})$, where $f(\mathbf{x})$ called (using neural network terminology) the "activation" at input point $\mathbf{x}$. In a Bayesian approach, all information about $f(\mathbf{x})$, when example data are known, is encoded in a posterior distribution of activations functions. The first ingredient to such an approach is the Likelihood of $f(\mathbf{x})$ which for noise-free classification and output label y is

Likelihood

$$p(y|f(\mathbf{x})) = \Theta(\, y\, f(\mathbf{x})) = \begin{cases} 1 & y\, f(\mathbf{x}) > 0 \\ 0 & y\, f(\mathbf{x}) < 0 \end{cases} \quad . \tag{17.1}$$

The second ingredient needed to form the posterior is the prior distribution over activations. A simple choice is a Gaussian process prior. This means that any finite set of function values

Gaussian Process prior

$$\mathbf{f} = (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_m)) \tag{17.2}$$

at arbitrary points $\mathbf{x}_1, \ldots, \mathbf{x}_m$ of the input space have a joint Gaussian distribution

$$p(\mathbf{f}) = \frac{1}{\sqrt{(2\pi)^m \det \mathbf{k}}} e^{-\frac{1}{2}(\mathbf{f}-\mathbf{m})^T \mathbf{k}^{-1} (\mathbf{f}-\mathbf{m})} \tag{17.3}$$

where $\mathbf{m} = (m(\mathbf{x}_1), \ldots, m(\mathbf{x}_m))$ is the mean and

$$\mathbf{k} \equiv E(\mathbf{f}\mathbf{f}^T) - \mathbf{m}\mathbf{m}^T \tag{17.4}$$

is the *covariance matrix* having elements

$$k(\mathbf{x}_i, \mathbf{x}_j), \quad i, j \in 1, \ldots, m \; . \tag{17.5}$$

The so-called *covariance function*, $k(\mathbf{x}, \mathbf{x}')$ is an explicit function of the pair of input points and determines correlations of activations at different points. A popular choice is the radial basis covariance function eq. (1.73), but any function that gives rise to a positive semidefinite covariance matrix can be used. The covariance function reflects our prior beliefs about the variability of the function

covariance function (kernel)

$f(\mathbf{x})$. The mean function $m(\mathbf{x})$ is usually set to a constant. The covariance function is completely equivalent to the Kernel function in the SVM approach as will be shown below.

17.2.1 Statistical Inference for Gaussian Processes

Given the training set

$$D_m = \{(\mathbf{x}_i, y_i) | i = 1, \ldots, m\}, \tag{17.6}$$

the inference task is to predict the correct label y on a new point $\mathbf{x}$. In the Bayesian framework, this is done by using the posterior distribution of $f(\mathbf{x})$ (which in the following will also be abbreviated by f). To calculate the posterior, the new activation is included in the prior: $p(\mathbf{f}, f(\mathbf{x}))$. The posterior is then given by

posterior

$$p(\mathbf{f}, f(\mathbf{x})|\mathbf{y}) = \frac{1}{p(\mathbf{y})} \underbrace{p(\mathbf{y}|\mathbf{f})}_{\text{Likelihood}} \underbrace{p(\mathbf{f}, f(\mathbf{x}))}_{\text{Prior}} , \tag{17.7}$$

where we have denoted the training set outputs by $\mathbf{y} = y_1, \ldots, y_m$ and the Likelihood of the training set activations is

$$p(\mathbf{y}|\mathbf{f}) = \prod_{i=1}^{m} p(y_i|f(\mathbf{x}_i)) = \prod_{i=1}^{m} \Theta(y_i\, f(\mathbf{x}_i)) \ . \tag{17.8}$$

Finally the normalization constant is

$$p(\mathbf{y}) = \int d\mathbf{f}\, p(\mathbf{y}|\mathbf{f})\ p(\mathbf{f}) \ . \tag{17.9}$$

The *predictive distribution* is

$$p(f(\mathbf{x})|\mathbf{y}) = \int d\mathbf{f}\, p(\mathbf{f}, f(\mathbf{x})|\mathbf{y}) \ . \tag{17.10}$$

Bayes optimal prediction

Using this distribution we can calculate the probability for output y: $p(y|\mathbf{y}) = \int df\ p(y|f)p(f|\mathbf{y})$. In the ideal case, (Bayes) optimal predictions are obtained by choosing the output with highest probability. For binary ± 1-classification, the Bayes classifier may be written as

$$y^{\text{Bayes}}(D_m, \mathbf{x}) = \operatorname{sgn} \int df\ p(f|\mathbf{y}) \operatorname{sgn} f \ . \tag{17.11}$$

The mean field approach–discussed in Section 17.6–aims at calculating an approximation to the Bayes classifier.

17.3 Modeling the Noise

So far we have only considered noise-free classification. In real situations, noise or ambiguities will almost always be present and are–in the Bayesian framework–at least conceptually straightforward to model.

input noise

We will consider two noise models: "input" (or additive) noise and output (multiplicative) noise. Input noise is defined as a random term added to the activation function in the likelihood:

$$p(y|f(\mathbf{x}), \xi(\mathbf{x})) = \Theta(y\,(f(\mathbf{x}) + \xi(\mathbf{x}))\,) \tag{17.12}$$

output noise

The output noise is flip noise, i.e.,

$$p(y|f(\mathbf{x}), \xi(\mathbf{x})) = \Theta(y\,\xi(\mathbf{x})\,f(\mathbf{x})) \tag{17.13}$$

where $\xi \in \{-1, +1\}$.
There are two ways to incorporate the input noise in the Gaussian Process framework: either to average it out by directly modifying the Likelihood according to

$$p(y|f) = \int d\xi\, p(y|f, \xi) p(\xi) \tag{17.14}$$

or to change variables to the "noisy" process $f + \xi$ with a modified prior and unchanged Likelihood eq. (17.1).

The simplest example is Gaussian noise with zero mean and variance v: The first approach gives the modified Likelihood

$$p(y|f) = \Phi\left(\frac{yf}{\sqrt{v}}\right), \tag{17.15}$$

where $\Phi(x) = \int_{-\infty}^{x} \frac{dy}{\sqrt{2\pi}} e^{-\frac{y^2}{2}}$ is an error-function. This Likelihood corresponds to *probit regression* [Neal, 1997]. In the second approach, we use the fact that the process $f + \xi$–due to the Gaussianity of the noise–is also a Gaussian process with the following covariance matrix

$$\mathbf{k}^{\text{noisy}} = \mathbf{E}\left[(\mathbf{f} + \boldsymbol{\xi})(\mathbf{f} + \boldsymbol{\xi})^T\right] - \mathbf{E}\left[\mathbf{f} + \boldsymbol{\xi}\right] \mathbf{E}\left[(\mathbf{f} + \boldsymbol{\xi})^T\right] = \mathbf{k} + v\mathbf{I}\,. \tag{17.16}$$

For output noise, we take an iid flip process which flips the classification label with a probability given by κ, thus

$$\begin{aligned} p(y|f) &= \sum_{\xi=\pm 1} p(\xi) p(y|f, \xi) \\ &= \kappa\Theta(-yf) + (1-\kappa)\Theta(yf) \\ &= \kappa + (1-2\kappa)\Theta(yf)\,. \end{aligned} \tag{17.17}$$

Such a noise process could model the effects of outliers, i.e., examples which are wrongly classified independently of the corresponding value of the activation function. Usually, we expect that the probability of a flip is small, when $f(\mathbf{x})$ is large and we have high confidence on the label. However, there may be some fraction of outliers in the data which may not be treated well by such a model. For those, we include the possibility that the probability of flip is independent of the location.

In the following, we will show 1. how SVM can be obtained from Gaussian processes with a modified (non-normalized) Likelihood and 2. the slack variable for SVM corresponds to the realization of the input noise ξ in the GP framework.

17.4 From Gaussian Processes to SVM

We will start by discussing the additive noise model and in the end of this section shortly consider the multiplicative noise model.

To obtain support vector machines from Gaussian processes, we may first look at the maximum a posteriori (MAP) values for activations and noise variables which can be obtained by maximizing the joint distribution

$$p(\mathbf{y}, \boldsymbol{\xi}, \mathbf{f}) = \prod_i \left[p(y_i|f_i, \xi_i)p(\xi_i)\right] p(\mathbf{f}) \ , \tag{17.18}$$

where we have suppressed the explicit $\mathbf{x}$ dependence. Equivalently, we may minimize the negative log posterior, $L = -\log p(\mathbf{y}, \boldsymbol{\xi}, \mathbf{f})$. Shifting the activation variables to a zero mean Gaussian process, i.e., $f(\mathbf{x}) \to f(\mathbf{x}) + m(\mathbf{x})$ with constant mean $m(\mathbf{x}) = b$ and enforcing the inequality constraints of the Likelihood $p(y|f, \xi) = \Theta(y(f+b+\xi))$ by non-negative Lagrange multipliers $\boldsymbol{\alpha}$, we arrive at

$$L = -\sum_i \log p(\xi_i) - \log p(\mathbf{f}) - \sum_i \alpha_i \left[y_i(f_i + b + \xi_i)\right] \ . \tag{17.19}$$

The MAP-parameters are obtained from the saddlepoint of L. A straightforward optimization $\frac{\partial L}{\partial f_i} = 0$ leads to the well known SVM expression

$$f_i^{\mathrm{SVM}} = \sum_j k_{ij} y_j \alpha_j \tag{17.20}$$

and the MAP prediction is given by

$$y^{\mathrm{SVM}}(\mathbf{x}) = \mathrm{sgn}(\sum_j k(\mathbf{x}, \mathbf{x}_j) y_j \alpha_j + b) \ . \tag{17.21}$$

Unfortunately, if the noise distribution has zero mean, the variation with respect to the other variables gives the trivial solution $\mathbf{f} = \boldsymbol{\alpha} = 0$. To obtain the SVM solution, a further *ad hoc* modification (equivalent to the introduction of a margin) is necessary. The final expression reads

$$L = -\sum_i \log p(\xi_i) - \log p(\mathbf{f}) - \sum_i \alpha_i \left[y_i(f_i + b + \xi_i) - 1\right] \ . \tag{17.22}$$

The expression for α_i and ξ_i obtained by a variation of this expression depends explicitly on the noise model. For Laplace noise $p(\xi) = \frac{C}{2}\exp(-C|\xi|)$, we obtain the Kuhn-Tucker conditions corresponding to the linear slack penalty $C\sum_i \xi_i$ (with $\xi_i \geq 0$) and Gaussian noise leads to the Kuhn-Tucker conditions corresponding to the quadratic slack penalty $\frac{1}{2v}\sum_i \xi_i^2$ [Cortes and Vapnik, 1995], Note that the mean of the Gaussian process b plays the role of the threshold (or bias) in the SVM framework.[1]

1. It is also possible to include a (e.g Gaussian) prior over b. The usual choice for SVM corresponds to a flat prior.

The *ad hoc* introduction of the extra margin destroys the probabilistic interpretation of the corresponding 'Likelihood' $p(y|f,\xi) = \Theta(y(f+b+\xi)-1)$ which does not correspond to a true probability, because it is not normalized, i.e., $\sum_{y=\pm 1} p(y|f,\xi) \leq 1$. Hence, a direct Bayesian probabilistic interpretation of SVM is not fully possible (at least in the simple MAP approach that we have sketched). So if we want to associate probabilities with output predictions, it is most natural to work in the Gaussian process framework (but see also Chapter 5). In practice however, it turns out that often the predictions made by both approaches are very similar when the same covariance function (kernel) and noise (slack) model are used.

It is *not* possible to follow the same scheme for the *output* noise realization $\xi = \pm 1$ because this leads to a combinatorial optimization problem which cannot be solved easily. Alternatively, one could use the Likelihood eq. (17.17) where the noise realization has been averaged out. However, eq. (17.17) is not a 0-1 probability corresponding to a simple inequality constraint that in the optimization may be enforced using a Lagrange multiplier. For inference with Gaussian processes–on the other hand–this is not a problem, since formally and practically, it is straightforward to deal with the Likelihood eq. (17.17) as we will see in Section 17.6.

17.5 Leave-One-Out Estimator

In this section, we derive an approximate leave-one-out (loo) estimator for the generalization error of the SVM-classifier. Although we do not know if our leave-one-out estimator can be cast into a bound on the true loo error (for bounds see [Jaakkola and Haussler, 1999b], Chapters 1 and 16), it seems to be at an excellent approximation (at least in the cases that we have applied it). Previously, we have given a derivation based on a limiting procedure of the TAP-mean field equations [Opper and Winther, 1999a]. The derivation given here is based on a linear response approach which is similar to the one derived by Wahba [1999b], however for a different loss function. For a similar approach in the framework of neural networks, see [Larsen and Hansen, 1996]. The approximation made in this approach is similar to an assumption which is also hidden in mean field theories: For systems which are composed of a large number of interacting degrees of freedom, a perturbation of a single one of them will change the remaining ones only slightly. To keep the derivation as simple as possible, we consider zero bias, $b = 0$. At the end of this section, we briefly sketch how to generalize the result to $b \neq 0$.

The basic idea is to calculate the change of the solution f_i for input i in response to removing example l. We will denote the solution at i without the lth example by $f_i^{\setminus l}$. Before and after the removal of example l, we have the following solutions

$$f_i = \sum_j k_{ij}\, y_j\, \alpha_j \tag{17.23}$$

$$f_i^{\setminus l} = \sum_{j \neq l} k_{ij}\, y_j\, \alpha_j^{\setminus l} \tag{17.24}$$

or

$$\delta f_i = \delta f_i^{\setminus l} \equiv f_i^{\setminus l} - f_i = \sum_{j \neq l} k_{ij}\, y_j\, \delta\alpha_j - k_{il}\, y_l\, \alpha_l \ . \tag{17.25}$$

There are two basic contributions to the change δf_i. The first term above is the indirect change due to the change of α_j in response to removing l and the second term is the direct contribution. The leave-one-out error is obtained as a simple error count

$$\epsilon_{\rm loo}^{\rm SVM} = \frac{1}{m} \sum_i \Theta\left(-y_i f_i^{\setminus i}\right) \ . \tag{17.26}$$

Unfortunately, the first term in eq. (17.25) cannot be calculated without making a specific approximation. The following derivation is for the SVM framework with linear slack penalty.

leave-one-out approximation

The Kuhn-Tucker conditions of SVM learning distinguishes between three different groups of examples. We make the assumption that example $j \in 1, \ldots, l-1, l+1, \ldots, m$, remains in the same group after retraining the SVM when example $l (\neq j)$ is removed. Explicitly,

1. Non-support vectors ($y_j f_j > 1$ and $\alpha_j = 0$), will remain non-support vectors: $\delta\alpha_j = 0$.

2. Margin support vectors ($y_j f_j = 1$ and $\alpha_j \in [0, C]$), will remain margin support vectors: $\delta f_j = 0$.

3. Misclassified patterns ($y_j f_j < 1$ and $\alpha_j = C$), will remain misclassified patterns: $\delta\alpha_j = 0$.

It is easy to construct a set of examples for which this assumption is not valid. We expect the approximation to be typically quite good when the number of support vectors is large because then upon removal of a support vector, the new solution will mainly be expressed as a (small) reorganization of the remaining margin support vectors. With this simplifying assumption, we may now solve eq. (17.25) in the form

$$\sum_{j \neq l}^{\rm mSV} k_{ij}\, y_j\, \delta\alpha_j - k_{il}\, y_l\, \alpha_l = 0 \tag{17.27}$$

to find $\delta\alpha_j$ for the margin support vectors (the non-support vectors and misclassified patterns are assumed to have $\delta\alpha_j = 0$).

It is necessary to consider explicitly the group to which the removed example belongs. We see immediately that if example l is a non-support vector then $\delta\alpha_j = 0$. If example l is a margin support vector, we get

$$\delta\alpha_i = \sum_{j \neq l}^{\rm mSV} \left[(\mathbf{k}_{\rm mSV}^{\setminus l})^{-1} \right]_{ij} k_{jl}\, y_l\, \alpha_l \ , \tag{17.28}$$

where $\mathbf{k}_{\mathrm{mSV}}^{\setminus l}$ is the covariance matrix of the margin support sector patterns excluding the lth pattern. Inserting the result in δf_i and setting $l = i$, we find

$$\delta f_i = \left\{ \sum_{j,j' \neq i}^{\mathrm{mSV}} k_{ij} \left[(\mathbf{k}_{\mathrm{mSV}}^{\setminus i})^{-1} \right]_{jj'} k_{j'i} - k_{ii} \right\} y_i \, \alpha_i = -\frac{1}{\left[\mathbf{k}_{\mathrm{mSV}}^{-1} \right]_{ii}} y_i \, \alpha_i \ . \tag{17.29}$$

In the last equality a matrix identity for the partitioned inverse matrix has been used.

For example l being a misclassified pattern, the sum in eq. (17.27) runs over all margin support vectors, thus

$$\delta \alpha_i = \sum_{j}^{\mathrm{mSV}} \left[\mathbf{k}_{\mathrm{mSV}}^{-1} \right]_{ij} k_{jl} \, y_l \, \alpha_l \ , \tag{17.30}$$

and

$$\delta f_i = \left\{ \sum_{j,j'}^{\mathrm{mSV}} k_{ij} \left[\mathbf{k}_{\mathrm{mSV}}^{-1} \right]_{jj'} k_{j'i} - k_{ii} \right\} y_i \, \alpha_i \ . \tag{17.31}$$

We see that the reaction δf_i is proportional to a direct change term through the factor α_i. We have now obtained the leave-one-out estimator eq. (17.26) for SVM with $y_i \, f_i^{\setminus i} = y_i \, f_i + y_i \, \delta f_i$ and δf_i given by eqs. (17.29) and (17.31) for respectively margin support vectors and misclassified patterns. Note that the sum over patterns will only run over support vectors since the reaction is estimated to be zero for non-support vectors.

One may argue that it is computationally expensive to invert $\mathbf{k}_{\mathrm{mSV}}$. However, we expect that the computational cost of this operation is comparable to finding the solution to the optimization problem since it–on top of identifying the support vectors–also requires the inverse of $\mathbf{k}_{\mathrm{SV}}$. This is also observed in simulations. Using this leave-one-out estimator is thus much cheaper than the exact leave-one-out estimate that requires running the algorithm N times (although each run will probably only take a few iterations if one uses an iterative learning scheme like the Adatron algorithm [Anlauf and Biehl, 1989] with the full training set solution as the starting point). Another possible way of decreasing the computational complexity of the estimator is to use methods in the spirit of the randomized GACV by Wahba [1999b].

These results may easily be generalized non-zero threshold: To include threshold f_i should be substituted with $f_i + b$. The Kuhn-Tucker condition for the margin support vectors therefore changes to $y_i(f_i + b_i) = 1$ which implies $\delta f_i = -\delta b$. E.g. for l being a margin support vector, we now have

$$\delta \alpha_i = \sum_{j \neq l}^{\mathrm{mSV}} \left[(\mathbf{k}_{\mathrm{mSV}}^{\setminus l})^{-1} \right]_{ij} (k_{jl} \, y_l \, \alpha_l - \delta b) \ . \tag{17.32}$$

The saddlepoint condition for b, $\frac{\partial L}{\partial b} = 0$, gives $\sum_i y_i \alpha_i = 0$. This condition implies $\sum_i^{\mathrm{mSV}} y_i \delta \alpha_i = 0$ which together with the expression for $\delta \alpha_i$ above determines δb.

17.6 Naive Mean Field Algorithm

The aim of the mean field approach is to compute an approximation to the Bayes prediction $y^{\text{Bayes}}(\mathbf{x}) = \text{sgn}\langle \text{sgn} f(\mathbf{x})\rangle$ for the GP classifier, where we have introduced the notation $\langle \ldots \rangle$ to denote a posterior average. We will only discuss a 'naive' mean field algorithm with the aim of stressing the similarities and differences between the SVM and Gaussian process approach. We will follow the derivation given in [Opper and Winther, 1999a] based on the so-called Callen identity [Parisi, 1988]. An independent derivation is given by Opper and Winther [1999b].

We will use the simplified prediction $y(\mathbf{x}) = \text{sgn}\langle f(\mathbf{x})\rangle$ which the Bayes classifier reduces to when the posterior is symmetric around its mean. We first give exact expressions for the posterior

$$\langle f(\mathbf{x})\rangle = \frac{1}{p(\mathbf{y})} \int d\mathbf{f} df \, f \, p(\mathbf{y}|\mathbf{f}) p(\mathbf{f}, f(\mathbf{x})) \; . \tag{17.33}$$

Using the following identity $f_j \, p(\mathbf{f}) = -\sum_i k(\mathbf{x}_j, \mathbf{x}_i) \frac{\partial}{\partial f_i} p(\mathbf{f})$ (or rather its extension to $p(\mathbf{f}, f)$), which is easily derived from (17.3) setting $\mathbf{m} = 0$, we can write

$$\langle f(\mathbf{x})\rangle = -\frac{1}{p(\mathbf{y})} \int d\mathbf{f} df \, p(\mathbf{y}|\mathbf{f}) \sum_i k(\mathbf{x}, \mathbf{x}_i) \frac{\partial}{\partial f_i} p(\mathbf{f}, f(\mathbf{x})) \tag{17.34}$$

We may now use integration by parts to shift the differentiation from the prior to the Likelihood:

$$\begin{aligned} \langle f(\mathbf{x})\rangle &= \sum_i k(\mathbf{x}, \mathbf{x}_i) \frac{1}{p(\mathbf{y})} \int d\mathbf{f} df \, p(\mathbf{f}, f(\mathbf{x})) \frac{\partial}{\partial f_i} p(\mathbf{y}|\mathbf{f}) \\ &= \sum_{i=1}^{m} k(\mathbf{x}, \mathbf{x}_i) \, y_i \, \alpha_i \; . \end{aligned} \tag{17.35}$$

Remarkably, this has the same form as the prediction of the SVM eq. (1.81). While for the SVM, the corresponding representation follows directly from the representer theorem of Kimeldorf and Wahba [1971], we can not use this argument for the mean field method, because (17.35) is not derived from minimizing a cost function. For the mean field approach, the "embedding strength" α_i of example i is given by

$$\alpha_i = \frac{y_i}{p(\mathbf{y})} \int d\mathbf{f} p(\mathbf{f}) \frac{\partial}{\partial f_i} p(\mathbf{y}|\mathbf{f}) \tag{17.36}$$

Note that the α_i's will always be non-negative when $p(y_i|f(\mathbf{x}_i))$ is an increasing function of $y_i f(\mathbf{x}_i)$.

We give now a mean field argument for the approximate computation of the α_i. There are different ways of defining a mean field theory. The present one has the advantage over other approaches [Opper and Winther, 1999a], that no matrix inversions are needed in the final algorithm. To proceed, auxiliary variables $\mathbf{t}$ are introduced using a standard Gaussian transformation

$$\alpha_i = \frac{y_i}{p(\mathbf{y})} \int \frac{d\mathbf{f} d\mathbf{t}}{(2\pi)^m} \exp\left(-\frac{1}{2}\mathbf{t}^T \mathbf{k} \mathbf{t} + i \mathbf{t}^T \mathbf{f}\right) \frac{\partial}{\partial f_i} p(\mathbf{y}|\mathbf{f}) \tag{17.37}$$

$$= \frac{y_i}{p(\mathbf{y})} \int \frac{d\mathbf{f}\, d\mathbf{t}}{(2\pi)^m} (-it_i) \exp\left(-\frac{1}{2}\mathbf{t}^T\mathbf{k}\mathbf{t} + i\mathbf{t}^T\mathbf{f}\right) p(\mathbf{y}|\mathbf{f}) = -iy_i\langle it_i\rangle \ ,$$

where the i *not* appearing as an index is the imaginary unit $i = \sqrt{-1}$. In the second equality integration by parts is applied. In the last equality the bracket is understood as a formal average over the joint complex measure of the variables $\mathbf{f}$ and $\mathbf{t}$. Next, we separate the integrations over f_i and t_i from the rest of the variables to get

$$\alpha_i = y_i \left\langle \frac{\int df_i dt_i \exp\left(-\frac{1}{2}k_{ii}(t_i)^2 + (-it_i)(\sum_{j\neq i} k_{ij}(-it_j) - f_i))\right) \frac{\partial p(y_i|f_i)}{\partial f_i}}{\int df_i dt_i \exp\left(-\frac{1}{2}k_{ii}(t_i)^2 + (-it_i)(\sum_{j\neq i} k_{ij}(-it_j) - f_i))\right) p(y_i|f_i)} \right\rangle \quad (17.38)$$

This identity can be proved by noting that the average over f_i and t_i in $\langle \ldots \rangle$ exactly cancels the denominator given us back the original expression for α_i.

We may now carry out the explicitly written integrals over f_i and t_i. Using the Likelihood for output noise eq. (17.17), we find

$$\begin{aligned}\alpha_i &= y_i \left\langle \frac{\int df_i \exp\left(-\frac{(f_i - \sum_{j\neq i} k_{ij}(-it_j))^2}{2k_{ii}}\right) \frac{\partial p(y_i|f_i)}{\partial f_i}}{\int df_i \exp\left(-\frac{(f_i - \sum_{j\neq i} k_{ij}(-it_j))^2}{2k_{ii}}\right) p(y_i|f_i)} \right\rangle \\ &= \frac{1}{\sqrt{k_{ii}}} \left\langle \frac{(1-2\kappa) D\left(\frac{\sum_{j\neq i} k_{ij}(-it_j)}{\sqrt{k_{ii}}}\right)}{\kappa + (1-2\kappa)\Phi\left(y_i \frac{\sum_{j\neq i} k_{ij}(-it_j)}{\sqrt{k_{ii}}}\right)} \right\rangle \ , \end{aligned} \quad (17.39)$$

"naive" mean field approximation

where $D(z) = e^{-z^2/2}/\sqrt{2\pi}$ is the Gaussian measure. So far everything is exact. The "naive" mean field approximation amounts to neglecting the fluctuations of the variable $\sum_{j\neq i} k_{ij}(-it_j)$ and substituting it with its expectation $\sum_{j\neq i} k_{ij}\langle -it_j\rangle = \sum_{j\neq i} k_{ij} y_j \alpha_j$. This corresponds to moving the expectation through the nonlinearities. One should however keep in mind, that the integrations are over a complex measure and that the t_j are not random variables in a strict sense. The result of this approximation is a self-consistent set of equations for $\alpha_i = -iy_i\langle t_i\rangle$. The explicit expression for α_i becomes

$$\alpha_i = \frac{1}{\sqrt{k_{ii}}} \frac{(1-2\kappa) D(z_i)}{\kappa + (1-2\kappa)\Phi(z_i)}, \qquad z_i = y_i \frac{\langle f_i\rangle - k_{ii} y_i \alpha_i}{\sqrt{k_{ii}}} \ . \quad (17.40)$$

In Figure 17.1, α_i is plotted as function of z_i (with $k_{ii} = 1$). The shape of the "embedding"-function depends crucially upon whether we model with or without output noise. For the noise-free case, $\kappa = 0$, α_i is a decreasing function of $y_i\langle f_i\rangle - k_{ii}\alpha_i = z_i\sqrt{k_{ii}}$ which may be thought of as a naive approximation to (y times) the activation for input i trained without the ith example. The result is intuitively appealing because it says that the harder it is to predict an example's label, the larger weight α_i it should have.[2] In the noisy case, α_i is a decreasing

2. In the more advanced TAP (named after Thouless, Anderson & Palmer) mean field theory z_i is proportional to the "unlearned" mean activation [Opper and Winther, 1999a].

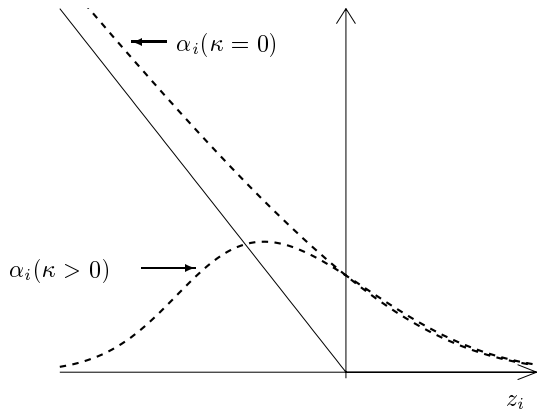


Figure 17.1 The "embedding strength" α_i plotted as a function of z_i with $k_{ii} = 1$.

function of z_i down to certain point at which the algorithm tends to consider the example as being corrupted by noise and consequently gives it a smaller weight. This illustrates the difference between flip noise and using the linear slack penalty for support vectors where the "hardest" patterns are given the largest weight, $\alpha_i = C$.

It is interesting to note that for the mean field algorithm α_i, in contrast to SVM, is an explicit function of other variables of the algorithm. The fact that the function is non-linear makes it impossible to solve the equations analytically and we have to resort to numerical methods. In Table 17.1, we give pseudo-code for a parallel iterative scheme for the solution of the mean field equations. An

Algorithm 17.1 : Naive mean field

Initialization:
Start from *tabula rasa*, $\boldsymbol{\alpha} := 0$.
Learning rate, $\eta := 0.05$.
Fault tolerance, ftol $:= 10^{-5}$.
Iterate:
while $\max_i |\delta\alpha_i|^2 >$ ftol **do:**
for all i:

$$\langle f_i \rangle := \sum_j k_{ij} y_j \alpha_j$$

$$\delta\alpha_i := \frac{1}{\sqrt{k_{ii}}} \frac{(1-2\kappa)D(z_i)}{\kappa + (1-2\kappa)\Phi(z_i)} - \alpha_i, \qquad z_i \equiv y_i \frac{\langle f_i \rangle - k_{ii} y_i \alpha_i}{\sqrt{k_{ii}}}$$

endfor
for all i**:**

$$\alpha_i := \alpha_i + \eta \delta\alpha_i$$

endwhile

important contributing factor to ensure (and to get fast) convergence is the use of an adaptable learning rate: We set $\eta := 1.1\eta$ if "the error" $\sum_i |\delta\alpha_i|^2$ decreases in

the update step and $\eta := \eta/2$ otherwise. Clearly, the algorithm does not converge for all values of the hyperparameters.[3] However, if the SVM has a solution for a certain choice of hyperparameters, the mean field algorithm will almost always converge to a solution and vice versa. The important question of how to tune the hyperparameters is discussed in the following.

leave-one-out estimator

For comparison, we also give the leave-one-out estimator for the naive mean field algorithm. It is derived from the mean field equations using linear response theory [Opper and Winther, 1999a] in completely the same fashion as the leave-one-out estimator for SVM

$$\epsilon_{\text{loo}}^{\text{naive}} = \frac{1}{m}\sum_{i}^{\text{SV}} \Theta\left(-y_i\langle f_i\rangle + \left\{\frac{1}{[(\mathbf{\Omega}+\mathbf{k})^{-1}]_{ii}} - \Omega_i\right\}\alpha_i\right) \ , \tag{17.41}$$

where $\mathbf{\Omega}$ is a diagonal matrix with elements

$$\Omega_i = k_{ii}\left(\frac{1}{y_i\alpha_i\langle f_i\rangle} - 1\right) \ . \tag{17.42}$$

We thus have the same basic structure as for the SVM estimator. However, this estimator requires the inversion of the full covariance matrix. In the next section, we will demonstrate on a benchmark dataset that the leave-one-out estimators are in very good agreement with the exact leave-one-out errors. This has also been observed previously on other benchmarks [Opper and Winther, 1999b,a]. We also show that despite the fact that this algorithm looks very different from SVM, the solution obtained and the performance is quite similar. The mean field approach will tend to produce smaller minimal margin, however we have not observed that this has any effect on performance.

17.7 Simulation Results

The two algorithms have been tested on the *Wisconsin breast cancer* dataset, which is a binary classification task (tumor is malignant or benign) based on 9 attributes, see, e.g., [Ster and Dobnikar, 1996]. We have removed the 16 examples with missing values and used standard preprocessing as to set the mean for every input equal to zero and the variance to unity across the dataset of 683 examples. The performance is–as in previous studies–accessed using 10-fold cross validation [Ster and Dobnikar, 1996].

For SVM, we used the parallel version of the Adatron algorithm of Anlauf and Biehl [1989] which, extended to general covariance functions, has turned out to

3. In Bayesian modeling, hyperparameters refer to "higher level" parameters which are not determined directly in the algorithm (in contrast to, e.g., $\boldsymbol{\alpha}$). The hyperparameters for this algorithm are the output flip probability κ, the input noise variance v and the input lengthscale(s) in the kernel, e.g., σ in the radial basis kernel eq. (1.73). The algorithm depends on the two latter hyperparameters only through the covariance matrix eq. (17.16).

be a fast iterative algorithm [Frieß et al., 1998]. For naive mean field theory, we solved the mean field equations using the iterative scheme described in the previous section.

We chose to work with the radial basis covariance function eq. (1.73). The Gaussian noise model is used in noisy process formulation thus adding the input noise variance v to the diagonal of the covariance matrix as in eq. (17.16). For the mean field algorithm, we have the additional output noise parameter κ. These two(three) parameters are chosen as to minimize the leave-one-out (loo) error for one of the 10 training sets by scanning through a number of parameter values. We found the values $\sigma^2 = 0.15/N$ and $v = 1.3$ for both algorithms and $\kappa = 0$. The true minimum is probably not found by this very rough procedure, however, the performance turned out to be quite insensitive to the choice of hyperparameters.

Since we use the training set to assess the performance through the 10-fold cross validation scheme, the loo estimate and test error are not independent. However, our main emphasis is not on generalization performance but rather on learning speed and on the precision of the loo estimators. The 10-fold cross validation error for respectively SVM and naive mean field theory is $\epsilon = 0.0307$ (21) and $\epsilon = 0.0293$ (20), where the numbers in parentheses indicate the number of misclassifications. The loo errors are $\epsilon_{\rm loo} = 0.0293$ and $\epsilon_{\rm loo} = 0.0270$. The more advanced TAP mean field algorithm [Opper and Winther, 1999b,a] finds a solution very similar to the one of the naive mean field algorithm. In another study using the SVM-algorithm, Frieß et al. [1998] find $\epsilon = 0.0052$. The difference may be due to a number of reasons: different splitting of the data set, different choice of hyperparameters, use of bias and/or handling of missing values. With other methods the following error rates are found: multi-layer neural networks $\epsilon = 0.034$, linear discriminant $\epsilon = 0.040$, RBF neural networks $\epsilon = 0.041$ and CART $\epsilon = 0.058$ [Ster and Dobnikar, 1996].

In Table 17.1, we compare the learning speed of the two algorithms–trained on one of the 10 training sets (with 614 examples)–both with and without evaluating the loo estimator (in CPU seconds on an Alpha 433au) and the number of iterations required to achieve the required precision, $\max_i |\delta\alpha_i|^2 < \text{ftol} = 10^{-5}$. We also compare the leave-one-out estimator $\epsilon_{\rm loo}$ with the exact loo estimator $\epsilon_{\rm loo}^{\rm exact}$ for both algorithms. In this case the loo estimators for both algorithms are in accordance with the exact values. Apart from the case where the value of σ is very small corresponding closely to a nearest-neighbor classifier, we have always observed that the leave-one-out estimators are very precise, deviating at most one classification from the correct value [Opper and Winther, 1999a].

Without evaluating the loo estimators, the naive mean field algorithm is about 4 times faster than the Adatron. With the leave-one-out estimator, the SVM is about 4 times faster than the naive mean field algorithm. This is due to the fact that for $\epsilon_{\rm loo}^{\rm SVM}$, eq. (17.26), we only need to invert the covariance matrix for the margin support vector examples, which in this example is 272-dimensional, whereas $\epsilon_{\rm loo}^{\rm naive}$, eq. (17.41) requires the inversion of the full covariance matrix (614-dimensional). If the linear slack penalty had been used, the number of support vectors would have been smaller and the advantage of using $\epsilon_{\rm loo}^{\rm SVM}$ would have been even greater.

Table 17.1 Results for the Wisconsin dataset.

Algorithm	$\epsilon_{\text{loo}}^{\text{exact}}$	ϵ_{loo}	CPU w. loo	CPU wo. loo	It.
SVM	0.0261	0.0261	5	4	195
Naive Mean Field	0.0293	0.0293	16	1	31

In Figure 17.2, we compare the solutions found by the two algorithms. The solutions for the "embedding strengths" α_i are quite similar. However, the small differences in embedding strength give rise to different distributions of margins. The mean field algorithm achieves both smaller and larger margins than SVM. We have also indicated which of the examples are predicted as wrongly classified by the loo estimators. Interestingly, these are almost exclusively all the examples with the highest α_i starting around the point where the α_i-curve's slope increases. This observation suggests that a heuristic cut-off for small α_i could be introduced to make the loo estimators faster without significantly deteriorating the quality of the estimators. Simple heuristics could be developed like, e.g., only considering the covariance matrix for the 10% of the examples with highest α_i, if one expects the error rate to be around 5%.

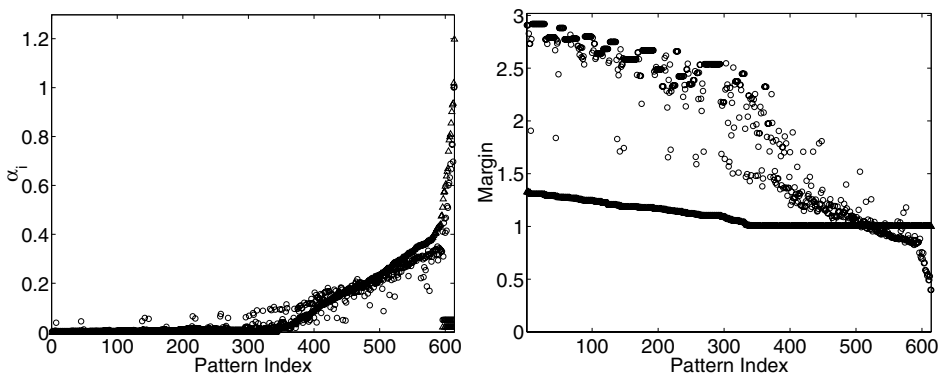


Figure 17.2 Left figure: The "embedding strengths" α_i for each example. The right figure: The margins $y_i f_i$ for SVM and $y_i \langle f_i \rangle$ for naive mean field theory (same ordering as the left plot). The triangles are for support vectors and circles are for naive mean field theory. They are sorted in ascending order according to their support vector α_i value and the naive mean field solution is rescaled to the length of the support vector solution. In the lower right corner of the left figure, it is indicated which examples contribute to the loo error.

17.8 Conclusion

This contribution discusses two aspects of classification with Gaussian Processes and Support Vector Machines (SVM). The first one deals with the relation between the two approaches. We show that the SVM can be derived as a maximum posterior prediction of a GP model. However, the corresponding likelihood is not normalized and a fully satisfactory probabilistic interpretation is not possible.

The second aspect deals with approximate approaches for treating two different computational problems arising in GP and SVM learning. We show how to derive an approximate leave-one-out estimator for the generalization error for SVM using linear response theory. This estimator requires only the inversion of the covariance matrix of the margin support vector examples. As the second problem we discuss the computation of the Bayes prediction for a GP classifier. We give a derivation of an algorithm based on a 'naive' mean field method. The leave-one-out estimator for this algorithm requires the inversion of the covariance matrix for the *whole* training set. This underlines a difference between SVM and GP which may have important practical consequences when working with large data sets: the GP solution lacks the sparseness property of SVM.

We have presented simulations for the Wisconsin breast cancer dataset, with the model hyperparameters determined by minimizing the approximate leave-one-out estimator. The performance of both algorithms was found to be very similar. The approximate leave-one-out estimators were in perfect agreement with the exact leave-one-out estimators.

An important problem for future research is to find efficient ways for tuning a larger number of hyperparameters in the kernel automatically. This will be necessary, e.g., in order to adapt the length-scales of the input components individually. The minimization of a leave-one-out estimator is only one possible technique for finding reasonable values for such parameters. Bayesian approaches to model selection such as the evidence (or MLII) method could be interesting alternatives [Berger, 1985, MacKay, 1992]. They are obviously well suited for the Bayesian GP approach. But they may also be interesting for an application to SVM. However, in order to implement such approaches properly, it will be necessary to understand the quantitative relations and differences between GP and SVM in more detail.

Acknowledgments

We are thankful to Thilo-Thomas Frieß, Pál Ruján, Sara A. Solla, Peter Sollich and Grace Wahba for discussions. This research is supported by the Swedish Foundation for Strategic Research.

V Beyond the Margin

18 Computing the Bayes Kernel Classifier

Pál Ruján
FB Physik and ICBM
Carl von Ossietzky Universität Oldenburg
Postfach 2503, D-26111 Oldenburg, Germany
rujan@neuro.uni-oldenburg.de
http://www.neuro.uni-oldenburg.de/∼rujan

Mario Marchand
SITE, University of Ottawa
Ottawa, K1N-6N5 Ontario, Canada
marchand@site.uottawa.ca
http://www.site.uottawa.ca/∼marchand

We present below a simple ray-tracing algorithm for estimating the Bayes classifier for a given class of parameterized kernels.

18.1 Introduction

Since the Editors of this book were kind enough to already write down the Introduction to this Chapter, we will concentrate here on a pictorial exposition of a simple but powerful algorithm for estimating the Bayes classifier. Two preliminary comments: if the finite set of examples under consideration is linearly separable - the examples belonging to two classes can be separated by a hyperplane - then using kernels instead of maximal margin perceptrons will probably decrease quite a bit the generalization error at the price of longer learning- and run times. This Chapter will show how to improve further the classification performance at the cost of even longer learning times.

Better learning takes more time

If the example set is not linearly separable, then there are two possible ways of getting rid of the intersection between the two classes convex hulls's. One is to partition each class in smaller pieces until the corresponding convex hulls do not overlap any longer. This idea leads to network growth algorithms as in [Ruján and Marchand, 1989, Marchand et al., 1989]. Another alternative is to embed the two sets in a high dimensional space where they must become linearly separable.

This can achieved by adding combinations of the already existing feautures to the input vectors. A particularly compact and elegant method was introduced in [Boser et al., 1992] and led to the kernel support vector machines. Both types of algorithms construct a network architecture depending on the training set and both use perceptrons as their basic units. The SVM approach is more successful due to its strong regularization properties and the available theoretical bounds on the expected structural risk. However, the training times are still quite long and, more importantly, running the classifier is slower than for usual feedforward networks.

18.2 A Simple Geometric Problem

Here is the main message of this Chapter in a two-dimensional nutshell. Consider a convex polygon $\mathcal{P}$ in 2D, defined through the set of side normal vectors $\mathbf{x}_i$: $\mathcal{P} = \{\mathbf{y} : (\mathbf{y} \cdot \mathbf{x}_i) \leq 1\}_{i=1}^{m}$, where $\mathbf{y} = (y_1, y_2)$. Given a direction $\mathbf{v}$ compute the line $(\mathbf{v} \cdot \mathbf{y}) = 1$ partitioning the polygon $\mathcal{P}$ into two parts of equal area $A_1 = A_2$ (see Figure 18.1). Call this line the Bayes decision line for direction $\mathbf{v}$.

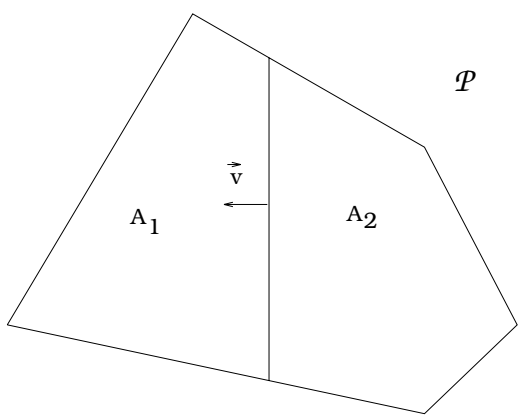


Figure 18.1 Partitioning a convex polyhedron in two equal volumes by a hyperplane with normal $\mathbf{v}$.

Now let us draw such Bayes lines in all possible directions, as shown in Figure 18.2. Contrary to our triangle preconditioned expectations, these lines do *not* intersect in one point. Hence, no matter how we choose a point inside the polygon $\mathcal{P}$, there will be directions along which the $\mathbf{v}$ oriented line will not partition $\mathcal{P}$ into two equal areas. The *Bayes point* is defined here as the point for which the direction-averaged squared area-difference is minimal.

The Bayes point

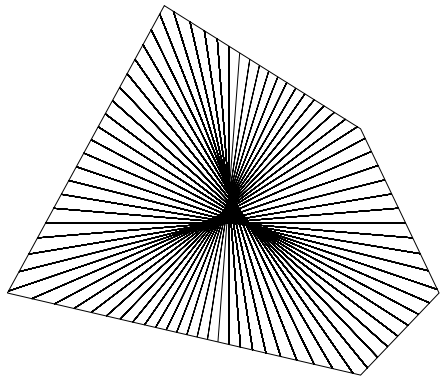


Figure 18.2 The set of Bayes-decision lines.

In general, it is rather difficult to compute the Bayes point. The convex polyhedra we will consider are defined through a set of inequalities, the vertices of the polyhedron are not known. Under such conditions one feasible approximation of the Bayes point is to compute the center of the largest inscribed circle or, better, the symmetry center of the largest area inscribed ellipse. As shown below, the center of the largest inscribed circle corresponds (in most cases) to the maximal margin perceptron. For strongly elongated polygons, a definitely better approach is to consider the center of mass of the polygon, as illustrated in Figure 18.3 Jumping suddenly into

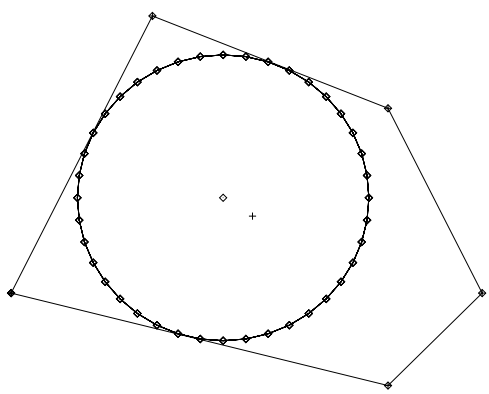


Figure 18.3 The largest inscribed circle and the center of mass (cross).

N-dimensions, the question is now how to sample effectively a high dimensional polyhedron in order to compute its center of mass. One possibility is to use Monte Carlo sampling [Neal, 1996] or sampling with pseudo-random sequences [Press et al., 1992].

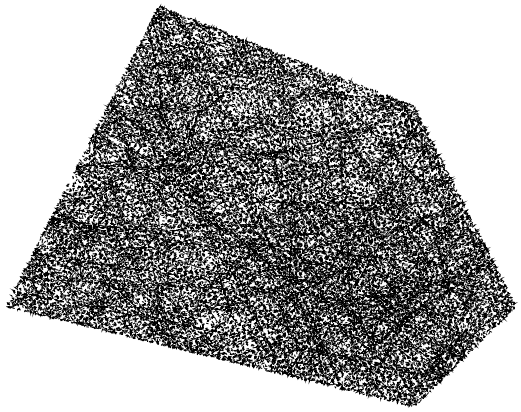


Figure 18.4 Trajectory (dashed line) after 1000 bounces.

Billiards

As suggested by one of us, [Ruján, 1997], another viable alternative is to use a ray-tracing method. The bounding polygon will be considered as a billiard table inside which a ball bounces elastically on the walls. With few exceptions corresponding to fully integrable or rational angle billiards, the so defined Hamiltonian dynamics will be ergodic, implying that a typical trajectory will homogeneously cover the polygon, as illustrated in Figure 18.4. It is known that the entropy of polygonal (and polyhedral) billiards vanish [Zemlyakov and Katok, 1975]. However, by excluding from the billiard a spherical region inside the polyhedron, one can make the dynamics hyperbolic [Bunimovich, 1979]. Such a billiard is somewhat similar to the Sinai-Lorenz billiard, which has strong mixing properties [Bunimovich and Sinai, 1980, Berry, 1981]. The absence of a general theorem, however, does not prevent us from using this algorithm. In the following, we will simply *assume* that a typical classification problem leads to ergodic dynamics and will sample accordingly the phase space. The presence of limit cycles, typical for fully integrable systems or KAM-tori in soft chaos can be - in principle - detected numerically.

18.3 The Maximal Margin Perceptron

Consider a set of N-dimensional m data points $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ belonging to two classes labeled by $\{y_1, \ldots, y_m\}$, $y_i = \pm 1$. We are seeking the plane normal $\mathbf{w}$ and two thresholds b_{+1}, b_{-1} such that

$$(\mathbf{w} \cdot \mathbf{x}_i) - b_{+1} > 0, \quad y_i = +1 \tag{18.1}$$

$$(\mathbf{w} \cdot \mathbf{x}_i) - b_{-1} < 0, \quad y_i = -1 \tag{18.2}$$

$$b_{+1} > b_{-1}$$

Maximal margin

This corresponds to finding two parallel hyperplanes passing between the two convex hulls of the positive and negative examples, respectively, such that their distance (the gap or margin) is maximal

$$G = \max_{\mathbf{w}} \frac{b_{+1} - b_{-1}}{(\mathbf{w} \cdot \mathbf{w})} \tag{18.3}$$

A set of linear inequalities...

This *primal* problem is the maximal margin or maximal "dead zone" perceptron [Vapnik, 1979, Lampert, 1969]. Eq. (18.1-18.2) and can be rewritten compactly as a set of linear inequalities

$$y_i(\mathbf{x}_i, 1)^T(\mathbf{w}, -b) \geq \Delta \geq 0, \qquad i = 1, 2, \cdots, m \tag{18.4}$$

...and its stability

where $b = \frac{b_{+1}+b_{-1}}{2}$ is the threshold of the maximal margin perceptron and $\Delta = \frac{b_{+1}-b_{-1}}{2}$ the *stability* of the set of linear inequalities (18.4). Note that in this notation the normal vector $(\mathbf{w}, -b)$ is also $N+1$-dimensional.

A two dimensional illustration of these concepts is shown in Figure 18.5.

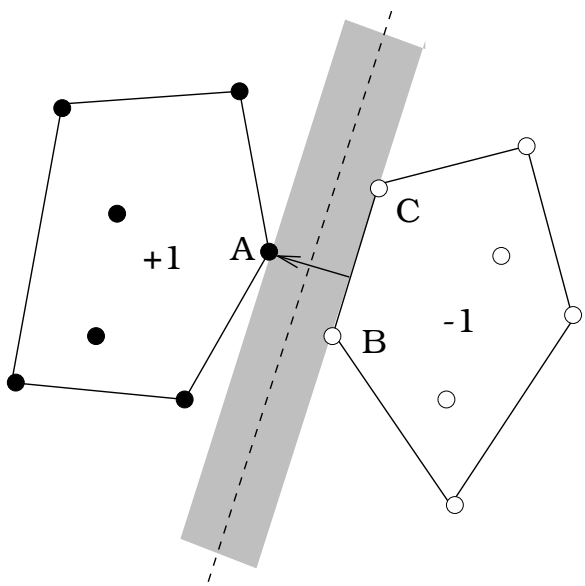


Figure 18.5 The maximal margin perceptron (once again!).

As shown by [Lampert, 1969] and geometrically evident from Figure 18.5, the *minimal connector* problem is *dual* to the maximal margin problem: find two convex combinations

$$\mathbf{X}_+ = \sum_{\{i:y_i=+1\}} \alpha_i^+ \mathbf{x}_i; \qquad \sum_{\{i:y_i=+1\}} \alpha_i^+ = 1; \quad \alpha_i^+ \geq 0 \tag{18.5}$$

$$\mathbf{X}_- = \sum_{\{i:y_i=-1\}} \alpha_i^- \mathbf{x}_i; \quad \sum_{\{i:y_i=-1\}} \alpha_i^- = 1; \quad \alpha_i^- \geq 0 \tag{18.6}$$

such that

$$L^2 = \min_{\{\alpha_i^+,\alpha_i^-\}} \|\mathbf{X}_+ - \mathbf{X}_-\|^2 \tag{18.7}$$

Minimal connector

The corresponding weight vector is given by $\mathbf{w}_{mm} = \mathbf{X}_+ - \mathbf{X}_-$. (18.7) together with the convexity constraints (18.5-18.6) defines a quadratic programming problem. In the mathematical programming literature the vertices A, B, and C in Figure 18.5 are called *active* constraints. Only the Lagrange multipliers α_i^+ and α_i^- corresponding to active vertices i are strictly positive, all others vanish. The active constraints satisfy the inequalities Eq. (18.1-18.2) as equalities. Furthermore, the gap G, Eq. (18.3) and the minimal connector's length L, Eq. (18.7) are equal only at optimality. Vapnik calls the active constraints *support vectors*, since the expression for $\mathbf{w}_{mm}$ (Eqs. (18.5-18.6)) involves two convex combination of the active constraints only.

Active constraints

A better geometric picture can be obtained by considering the linear conjugate vector space, so that Eq. (18.4) describes now hyperplanes whose normals are given by $\mathbf{z}_i = y_i(\mathbf{x}_i, 1)$. In this space all $\mathbf{W} = (\mathbf{w}, -b)$ vectors satisfying the linear inequalities Eq. (18.4) lie within the convex cone shown in Figure 18.6. The ray corresponds to the direction of the maximal margin perceptron. The point at which it intersects the unit sphere is at distance

The version space

$$d_i^{mm} = \left| \frac{(\mathbf{w}_{mm} \cdot \mathbf{x}_i)}{\sqrt{(\mathbf{x}_i \cdot \mathbf{x}_i)}} - b \right| = \Delta \tag{18.8}$$

from the active example $\mathbf{x}_i$. Hence, *if all active examples have the same length*, the maximal margin perceptron corresponds to the center of the largest sphere inscribed into the spherical polyhedron defined by the intersection of the unit sphere with the version space polyhedral cone.

The max margin perceptron as largest inscribed sphere

18.4 The Bayes Perceptron

Given a set of data $\mathbf{z} = \{\mathbf{z}_i\}_{i=1}^m$ and an unknow example $\mathbf{x}$, the optimal prediction for the corresponding class label $\mathbf{y}$ for a squared error loss is given by (see for example [Neal, 1996]):

$$f(\mathbf{x}) = \int g(\mathbf{x}, \mathbf{W}) P(\mathbf{W}|\mathbf{z}) d\mathbf{W} \tag{18.9}$$

where g is the perceptron output-function, $g(\mathbf{x}, \mathbf{W}) = \text{sgn}\{(\mathbf{x} \cdot \mathbf{w}) - b\}$ and $P(\mathbf{W}|\mathbf{z})$ the posterior network distribution. As usual, the Bayes identity

$$P(\mathbf{W}|\mathbf{z}) = \frac{P(\mathbf{z}|\mathbf{W})P(\mathbf{W})}{P(\mathbf{z})} \tag{18.10}$$

relates the posterior distribution to the prior $P(\mathbf{W})$. It was shown by [Watkin, 1993] that for an *iid* posterior, the Bayes classifier Eq. (18.9) can be represented by

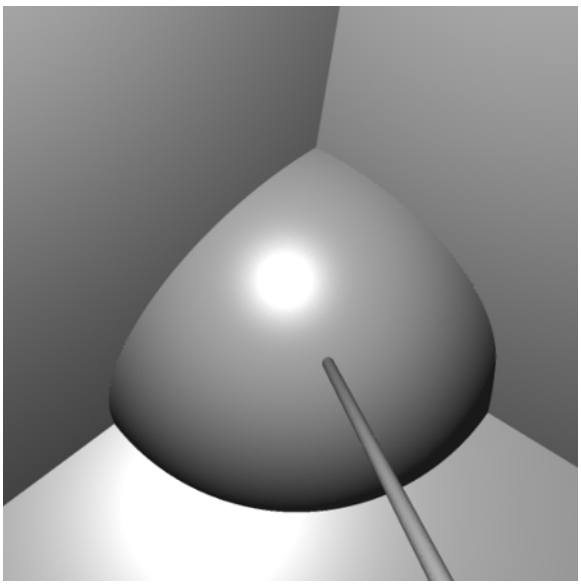


Figure 18.6 The version space is a convex polyhedral cone. The upper left plane corresponds to the active vertex A, the one upper right to the vertex B, and the lower one to the vertex C in Figure 18.5. The version space is restricted to the intersection between the unit sphere and the convex cone by the normalization constant $(\mathbf{W}_i \cdot \mathbf{W}_i) \equiv 1$. $i = 1, \ldots, m$. The ray corresponds to the direction of the maximal margin perceptron.

Bayes point = Bayes perceptron

a single perceptron corresponding to the center of mass of the version space:

$$\lim_{N\to\infty, M\to\infty, \frac{m}{N}=const} f(\mathbf{x}) = g(\mathbf{x}, \mathbf{W}^*) \tag{18.11}$$

where

$$\mathbf{W}^* = \frac{1}{\text{vol}\,(\mathbf{W})} \int_{\mathbf{W}\in V} \mathbf{W} dV \tag{18.12}$$

Hence, the center of mass of the intersection of the version space polyhedral cone with the unit sphere defines the Bayes perceptron parameters as long as the posterior is uniform. It is, however, not difficult to generalize the sampling algorithm presented below to nonuniform posteriors. As described in Chapter 1, the Bayes point is the center of mass of a polyhedral hypersurface whose local mass density is proportional to the posterior distribution (18.10). Under the ergodicity assumption the ray-tracing algorithm provides a collection of homogeneously distributed sampling points. The center of mass can be then estimated by a weighed average of these vectors, where the weights are proportional to the posterior probability density. Before describing the implementation of the billiard algorithm in more detail, we consider the generalization of these ideas to SVM.

18.5 The Kernel-Billiard

As explained in the introduction to this book (Chapter 1), the support vector machines are somewhat similar to the method of potential functions and "boosting" architectures. From a physicist's point of view, however, the support vector machines (SVM) operate on the quantum probability amplitude (wave-packets) $\mathbf{\Phi}$ with $\mathcal{L}^2$ metric, while the method of potential functions and the boosting on a classical probability density with $\mathcal{L}^1$ metric. To see this consider Figure 1.4 in Chapter 1: the SVM architecture acts as an *operator* on the input $\mathbf{\Phi}(\mathbf{x})$ feature vector, while the method of potential functions propagates directly the input vector through the set of real functions $\mathbf{\Phi}(\mathbf{x}_i)$, the dot-product layer is missing. The boosting architecture requires, in addition, the weights connected to the output unit to be convex coefficients. Hence, they can be interpreted as the probability that the "hypothesis" function $\mathbf{\Phi}_i$ is correct.

In order to generalize the perceptron learning method described above to kernels, one must rewrite the algorithm in terms of dot-products of the form $q_{ij} = (\mathbf{x}_i \cdot \mathbf{x}_j)$. If this is possible, one makes the substitution $q_{ij} \leftarrow (\mathbf{\Phi}(\mathbf{x}_i) \cdot \mathbf{\Phi}(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j)$, where $k(\mathbf{x}, \mathbf{y})$ is a Mercer-kernel. The main trick is thus to substitute this positive definite kernel for all scalar products. The explicit form of $\mathbf{\Phi}_i \equiv \mathbf{\Phi}(\mathbf{x}_i)$ is not needed.

Change in the dot-product

18.5.1 The Flipper Algorithm

Algorithm 18.1 : Flipper [Ruján, 1997]

1. Initialize center of mass vector, counters, etc., normalize example vectors according to Eq. (18.19).
2. Find a feasible solution $\mathbf{W}$ inside the version space,
3. Generate a random unit direction vector $\mathbf{V}$ in version space,
4. For iteration $n_B < N_{max}$ max-iterations, (flight time $\tau < \tau_{max}$ max-time)
   - compute flight-times to all bounding planes (m^2 dot-products $\sim$ kernel evaluations),
   - compute plane-index of the next bounce (corresponds to shortest positive flight time),
   - compute new position in version space $\mathbf{W}'$ and store it in center of mass vector,
   - compute the reflected direction vector $\mathbf{V}'$,
5. Test conditions : go back to 3) if escaped to ∞ or exit at N_{max}

Therefore, the typical number of operations is $O(M^2 N_{max})$ kernel evaluations. To find a feasible solution use any perceptron learning algorithm or the *light-trap method* described below. Figure 18.7 illustrates the main step of the algorithm. We compute the flight times for all planes and choose the smallest positive one. This will be the plane first hit by the billiard trajectory. We move the ball to the

Elastic scattering

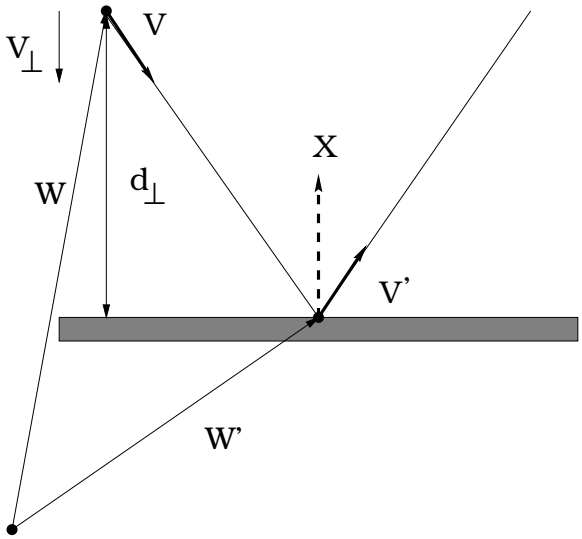


Figure 18.7 Bouncing on a plane. The flight time from point $\mathbf{W}$ to the plane can be computed from the scalar products $d_\perp = (\mathbf{W}\cdot\mathbf{X})$ and $V_\perp = (\mathbf{V}\cdot\mathbf{X})$ as $\tau = -d_\perp/V_\perp$. The collision takes place at $\mathbf{W}' = \mathbf{W} + \tau\mathbf{V}$ and the new direction is $\mathbf{V}' = \mathbf{V} + 2V_\perp\mathbf{X}$.

new position and reflect the direction of flight. The arc between the old and the new position is added according to the rules derived below to the center of mass estimate. Since the version space is a polyhedral cone, the trajectory will escape sometimes to infinity. In such cases we restart it from the actual estimated center of mass. Note the analogy to the two-dimensional geometrical problem mentioned above.

How many bounces N_B do we need before the estimated center of mass vector will be within an ε distance from the true center of mass with probability $1-\eta$? *If* the generated series of vectors $\mathbf{W}$ are identically and independently generated from the posterior distribution, then, as shown in Appendix A, applying Hoeffding's inequality results in

$$N_B > \Lambda \frac{2m}{\varepsilon^2} \ln \frac{2m}{\eta} \tag{18.13}$$

Convergence estimate

where m is the number of training vectors and for simplicity we assume that each component of the training vectors $\mathbf{x}$ lies in the interval $[-1, 1]$. The Hoeffding's estimate is multiplied by the correlation length, Λ.

Let us describe a given trajectory by the sequence of hyperplane indices hit by the billiard ball. The correlation length is the rate at which the correlation between these symbols decreases. Consider *two* trajectories started from neighboring points in slightly different directions. It seems reasonable to assume that the two trajectories will become fully uncorrelated once the two balls start to bounce on different

planes. Hence, Λ equals the average number of bounces after which two trajectories originally close in phase space bounce for the first time on different planes. The very important question whether Λ is constant or some function of m is still open.

Note that the best known algorithm for estimating the center of mass of a convex polyhedron uses the largest volume inscribed ellipsoid and apart logarithmic terms scales with $O(m^{3.5})$ [Khachiyan and Todd, 1993]. If Λ grows slower than $m^{\frac{1}{2}}$, than the flipper algorithm is faster than the Khachiyan – Todd algorithm. By introducing slight changes in our algorithm we can enhance the mixing properties of the billiard and thus optimize Λ. A simple solution is, for instance, to introduce a reflecting sphere around the (estimated) center of mass, lying completely inside the polyhedron. Another possibility is to use a random, nonlinear scattering angle function simulating a dispersing boundary. Such a dynamics would dissipate energy but leads to better mixing properties. This interesting topic will be addressed elsewhere.

When generalizing the billiard algorithm to kernel methods, we have first to show that the center of mass of the now very high dimensional space lies in the span defined by the example set. Let $\mathbf{\Phi}_i \equiv \mathbf{\Phi}(\mathbf{x}_i)$ denote in feature space the image of the training example $\mathbf{x}_i$. The version space $\mathcal{V}$ is defined to be the following set of weight vectors:

$$\mathcal{V} = \{\mathbf{w} : (\mathbf{w} \cdot \mathbf{\Phi}_i) > 0 \text{ for } i = 1, \ldots, m\} \tag{18.14}$$

where m denotes the number of training examples.

Any vector $\mathbf{w}$ lying in the version space can be written as $\mathbf{w} = \mathbf{w}_{\|} + \mathbf{w}_{\perp}$ where $\mathbf{w}_{\|}$ lies in the space spanned by $\{\mathbf{\Phi}_i\}_{i=1}^m$:

$$\exists\ \alpha_1, \ldots, \alpha_m : \mathbf{w}_{\|} = \sum_{i=1}^{m} \alpha_i \mathbf{\Phi}_i \tag{18.15}$$

and $\mathbf{w}_{\perp}$ lies in the orthogonal complement of that space (with respect to $\mathcal{V}$):

$$(\mathbf{w}_{\perp} \cdot \mathbf{\Phi}_i) = 0\ \forall i = 1, \ldots, m \tag{18.16}$$

Hence, $(\mathbf{w}_{\|} + \mathbf{w}_{\perp}) \in \mathcal{V}$ if and only if $(\mathbf{w}_{\|} - \mathbf{w}_{\perp}) \in \mathcal{V}$. For each $\mathbf{w}$ the $\mathbf{w}_{\perp}$ components cancel and the center of mass $\mathbf{w}^*$:

$$\mathbf{w}^* = \frac{1}{\text{vol}\,(\mathcal{V})} \int_{\mathcal{V}} \mathbf{w} dw \tag{18.17}$$

Center of mass is in examples's span

lies in the space spanned by $\{\mathbf{\Phi}_i\}_{i=1}^m$.

This small Lemma implies that instead of choosing a general direction in feature space we can restrict ourselves to a vector lying in the subspace spanned by the example vectors:

$$\mathbf{V} = \left(\sum_{i=1}^{m} \beta_n \mathbf{\Phi}_i, v_0 \right) \tag{18.18}$$

subject to the normalization condition

$$(\mathbf{V} \cdot \mathbf{V}) = \sum_{i=1}^{m} \sum_{j=1}^{m} \beta_i \beta_j k(\mathbf{x}_i, \mathbf{x}_j) + v_0^2 = 1 \tag{18.19}$$

The formulation of the flipper algorithm ensures that all subsequent direction vectors will also lie in the linear span. As usual, we expand also the weight vector as

$$\mathbf{W} = \left(\sum_{i=1}^{m} \alpha_i \mathbf{\Phi}_i, -b \right) \tag{18.20}$$

The normal vectors of the version space polyhedron boundaries are given in this description by

$$\mathbf{Z}_p = \frac{1}{\sqrt{k(\mathbf{x}_p, \mathbf{x}_p) + 1}} (y_p \alpha_p \mathbf{\Phi}_p, y_p) \tag{18.21}$$

where $p \in P$ denotes the subset of the extended example vectors (hyperplanes), which bounds from inside the zero-error feature space. Support vectors as defined by the maximal margin algorithm (MMSV) are related to those training examples which touch the largest inscribed hypersphere. Support vectors as defined in a Bayesian context correspond to hyperplanes bounding the zero-error feature space (BSV). Obviously, $MMSV \subseteq BSV \subseteq \mathbf{X}$.

Feature space support vectors

In general, the vectors $\mathbf{\Phi}_i,\ i = 1, \ldots, m$ are neither orthogonal, nor linearly independent. Therefore, the expansions (18.18)-(18.20) are not unique. In addition, some of the example hyperplanes might (will!) lie outside the version space polyhedral cone and therefore cannot contribute to the center of mass we are seeking. *Strictly speaking, the expansions in Eqs. (18.18) - (18.21) should be only in terms of the $p \in P$ support vectors.* This set is unknown, however, at the beginning of the algorithm. This *ambiguity*, together with the problem of a drifting trajectory (discussed below in more detail), are specific to the kernel method.

As explained in Figure 18.7, if we bounce on the j-th example hyperplane

$$V_j^{\perp} = \sum_{i=1}^{m} y_j \beta_i k(\mathbf{x}_i, \mathbf{x}_j) + v_0 y_j \tag{18.22}$$

and, similarly

$$W_j^{\perp} = \sum_{i=1}^{m} y_j \alpha_i k(\mathbf{x}_i, \mathbf{x}_j) - b y_j \tag{18.23}$$

Flight times

Once the example index j^* corresponding to the smallest positive flight time $\tau_{\min}$,

$$j^* = \underset{j, \tau_{\min} > 0}{\operatorname{argmin}}\ \tau_{\min}, \quad \tau_{\min} = -\frac{W_j^{\perp}}{V_j^{\perp}} \tag{18.24}$$

has been calculated, the update rules for $\mathbf{V}$ and $\mathbf{W}$ are

$$\beta_i \leftarrow \beta_i - 2 y_{j^*} \mathbf{V}^{\perp} \delta_{i,j^*} \tag{18.25}$$

$$v_0 \leftarrow v_0 - 2y_{j^*}\mathbf{V}^{\perp}$$

and

$$\begin{aligned} \alpha_i &\leftarrow \alpha_i + \tau_{\min}\beta_i \\ b &\leftarrow b + \tau_{\min} v_0 \end{aligned} \tag{18.26}$$

Center of mass update

In order to update the center of mass vector we define the sum of two unit vectors $\mathbf{a}$ and $\mathbf{b}$ as follows (see Figure 18.8). Each vector has an associated weight, denoted by ρ_a and ρ_b, respectively. The sum of the two vectors is $\mathbf{c}$, with weight $\rho_c = \rho_a + \rho_b$. If we choose the vector $\mathbf{a}$ as the unit vector in direction x, then, according to Figure 18.8, the coordinates of the three vectors are $\mathbf{a} = (1, 0)$, $\mathbf{b} = (\cos\phi, \sin\phi)$, and $\mathbf{c} = (\cos\phi_1, \sin\phi_1)$, respectively. Note that $\cos\phi = (\mathbf{a} \cdot \mathbf{b})$. We now require that the angle of the resultant vector $\mathbf{c}$ equals $\phi_1 = \frac{\rho_b}{\rho_c}\phi$, as when adding parallel forces.

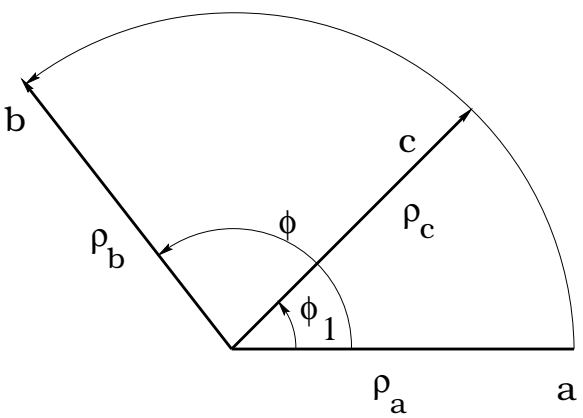


Figure 18.8 Adding two vectors lying on the unit hypersphere. The weights are denoted by ρ and $\mathbf{c} = \mathbf{a} + \mathbf{b}$ implies $\rho_c = \rho_a + \rho_b$.

This leads to the following addition rule

$$\mathbf{c} = \cos\left(\frac{\phi\rho_b}{\rho_a + \rho_b}\right)\mathbf{a} + \frac{\sin\left(\frac{\phi\rho_b}{\rho_a+\rho_b}\right)}{\sin(\phi)}[\mathbf{b} - \cos(\phi)\mathbf{a}] \tag{18.27}$$

where $\cos\phi = \mathbf{ab} \leftarrow \mathbf{k}(\mathbf{a}, \mathbf{b})$. As explained in [Rujan, 1997], we can add recursively using Eq. (18.27) the middle points of the arcs between the two bouncing points $\mathbf{W}$ and $\mathbf{W}'$, with a weight given by the length of the arc, which is proportional to the angle ϕ. If the new direction points towards the free space, $\tau_{\min} = \infty$, we restart the billiard at the actual center of mass in a randomly chosen direction. If the number of bounces and the dimension are large enough, which is almost always the case, then it is enough to add the bouncing points with the weight determined by the posterior probability density.

Now consider the case when the number of BSV's is less than m. This means that in the example set there are some hyperplanes which can not contribute to the center of mass. The simplest such example occurs when the ball bounces for a long time between two given planes while moving (drifting) along with a constant speed, while the other examples are inactive. According to the update rules Eqs. (18.25-18.26), *all* examples contribute to both $\mathbf{w}$ and the threshold b! However, when using the center of mass for classification the contributions from inactive examples cancel out.

18.5.2 The Light-Trap Algorithm

We can extend the main idea behind the billiard dynamics in several ways. One variant is the *light-trap* algorithm, which can be used to generate a feasible solution. Instead of the full reflecting mirror in Figure 18.7, consider a half-reflecting mirror, schematically shown in Figure 18.9

Trapping billiards

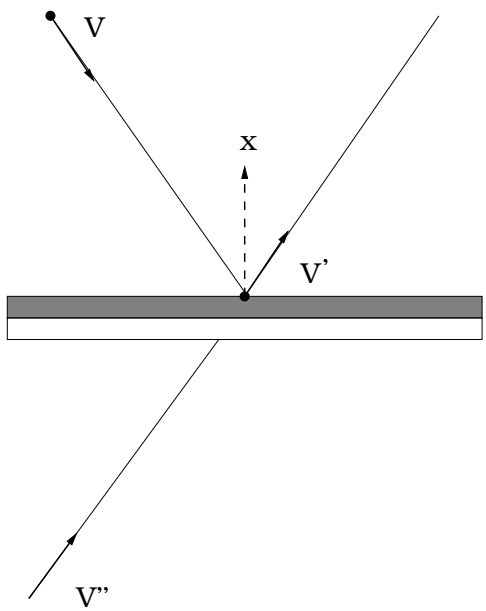


Figure 18.9 Bouncing on a semi-transparent plane. If the "light" comes from above ($\mathbf{V}_\perp < 0$) the trajectory is elastically reflected. If, however, the light shines from below, ($\mathbf{V}_\perp > 0$), the trajectory is allowed to pass through the mirror.

Therefore, if we start the trajectory at a point making few errors in version space, any time we encounter a non satisfied linear inequality we will "pass" through it, reducing by one the number of errors. Eventually, the trajectory will be trapped in the version space with zero errors. This algorithm is particularly simple to

implement for the Gaussian kernel

$$k(\mathbf{x},\mathbf{y}) = \mathrm{e}^{\frac{-\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}} \tag{18.28}$$

Gauss kernels

where the starting point corresponds to the limit $\sigma \to 0$. In this limit all $\mathbf{\Phi}_i$ vectors are pairwise orthogonal and $\mathbf{X}_-$, $\mathbf{X}_+$ are given by the center of mass of negative and positive example vectors in feature space: $\alpha_i = 1/m_-$ for negative and $\alpha_i = 1/m_+$ for positive examples, respectively. $m_{\pm 1}$ is the number of positive (negative) training examples.

This is nothing else than a Bayesian decision for fully uncorrelated patterns, the only useful information being the occurrence frequency of the two classes. This solution is a good starting point, making relatively few errors also for finite σ values. This idea can be easily implemented and provides in this context a new perceptron-learning method, thus making the billiard method self contained.

18.5.3 The Soft Billiard

Another natural extension is to "soften" the billiard, by allowing for one or more errors, as illustrated in the Figure 18.10

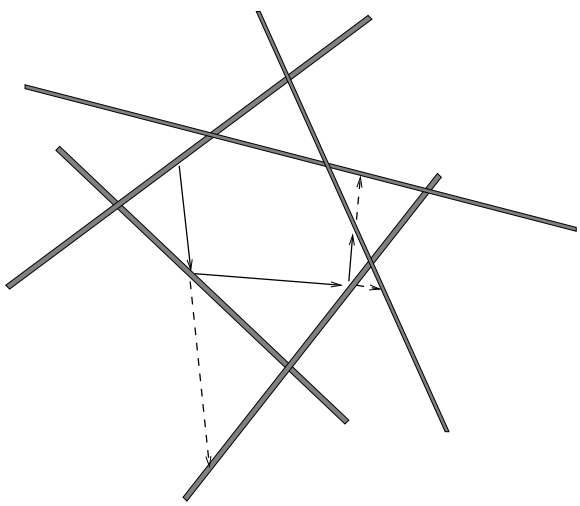


Figure 18.10 Projecting a trajectory beyond the zero-error version space.

Soft billiards

The "onion-algorithm" *sorts* the positive flight times in increasing order. While the shortest flight time corresponds to the boundary of the zero-error feature space polyhedron, we can sample now the position this ray would have on the second (one-error version space), third (two-error version space), etc., bounce. Therefore, while keeping the trajectory inside the zero-error cone, we generate at a minimal additional cost simultaneously estimates of the zero, one-error, two-errors, etc.

version spaces. By keeping track of the original labels of the planes the trajectory is bouncing upon, we produce a set of "center-of-mass" vectors corresponding to a given (n_-, n_+) number of negative and positive errors, respectively. After convergence, we can try different ways of weighting these results without having to rerun the sampling procedure again.

This method provides a powerful tool of searching for good "mass-centers" in addition to changing the kernel parameter σ. It has also the same drawback, namely it requires a careful testing according to the "k out on m"-estimate protocol.

18.6 Numerical Tests

We did not yet perform exhaustive tests on the SVM variant of these algorithms. While in the process of editing this manuscript we received an article [Herbrich et al., 1999a] where numerical results obtained on a large set of data strongly support our conclusions. For illustration purposes we present below results obtained for normal perceptrons, were also analytic results are available [Opper and Haussler, 1991]. The training examples have been generated at random and classified according to a randomly chosen but fixed "teacher" perceptron. Figure 18.11 shows the theoretical Bayes learning curve. The experimental results were obtained using the billiard algorithm and represents an average over 10 different runs.

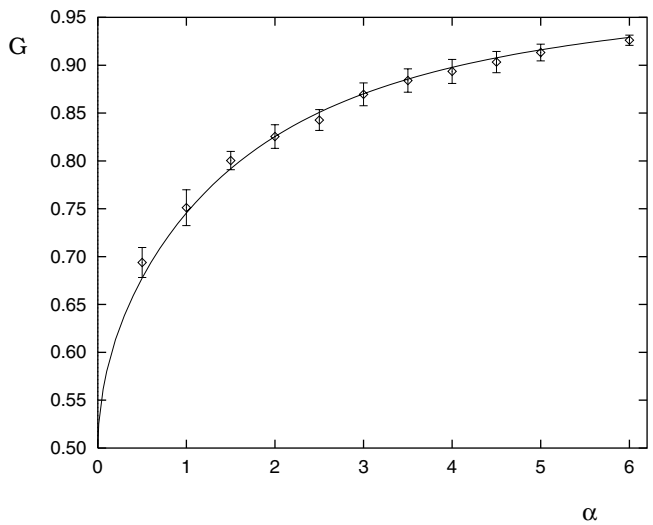


Figure 18.11 The learning curve (generalization probability) as a function of $\alpha = \frac{M}{D}$, where M is the number of randomly generated examples and $D = 100$ the input dimension. The continuous curve is the Bayes result of Opper and Haussler, the points with error bars represent billiard results.

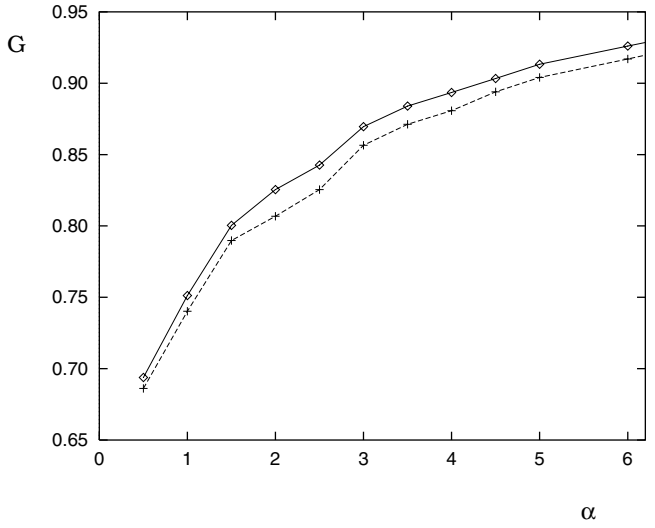


Figure 18.12 Comparison between average generalization values obtained with the maximal margin perceptron and the flipper algorithm, respectively. Same parameter values as in Figure 18.11.

Figure 18.11 shows a comparison between the average generalization error of the maximal margin perceptron (lower curve) vs. the billiard results. Not shown is the fact that in each single run the maximal margin perceptron was worse than the billiard result. We present below the results obtained with the Gaussian kernel Eq. (18.28) for two real-life data sets. In Figure 18.13 are the sonar data [Gorman and Sejnowsky, 1988], split into two sets according to the aperture angle. Note that different versions of this split are in circulation. Our split can be found in P.R.'s www-home page (http://www.neuro.uni-oldenburg/~rujan)

These results show that using the onion-algorithm one can obtain very good, stable results for a rather wide range of σ values.

We also tested our algorithms on the Wisconsin breast cancer data collected by W. H. Wolberg (Breast Cancer Database, University of Wisconsin Hospitals, Madison). After removing all incomplete examples we are left with 673 cases. Each has 9 attributes and 2 classes. Using the leave-10-out cross-validation method we obtain for $\sigma = 1.75$ 25 errors for the maximal margin SVM and 23 errors by the billiard algorithm (out of 670 tests).

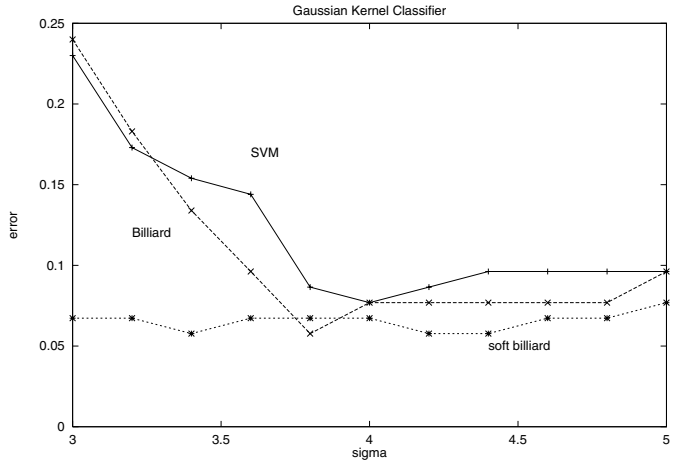


Figure 18.13 The aspect-angle dependent sonar data split: comparison between the generalization error obtained with the maximal margin SVM, the flipper, and the onion algorithms, respectively. Note that different data splits are available in the literature.

18.7 Conclusions

Although we have not yet tested in great detail the billiard algorithm for SVM's, we believe that this algorithm and its variants provide good estimates for the Bayesian kernel classifiers. Another interesting observation is that the best results were obtained just before the billiard dynamics "closed." In general, for small values of the kernel parameter σ, the trajectories are rather short, the ball escapes often to ∞. As σ increases the trajectory length increases as well. A phase-transition like change happens for even larger values of σ: the length of the trajectory seems to diverge, the billiard is closed.

Further theoretical work is necessary for determining correctly the typical correlation length Λ in Eq. (18.13). If it turns out that Λ does not depend on the dimension (number of training examples), then the estimate (18.13) would suggest that the billiard method is superior to any known algorithm for solving convex programming problems.

Ideal gas of balls

In the present implementation the billiard based algorithms are slow in comparison to the QP algorithms, for instance. However, since they can be run in parallel on different processors without having to exchange a lot of data, they are well suited for massively parallel processing.

err	+0	+1	+2	+3	+4	+5
-0	8 (3,5)	9 (1,8)	13 (1,12)	18 (1,17)	20 (0,20)	21 (0,21)
-1	10 (5,5)	7 (2,5)	8 (1, 7)	8 (1, 7)	9 (1, 8)	11 (1,10)
-2	10 (5,5)	8 (3,5)	8 (2, 6)	6 (1, 5)	10 (1, 9)	16 (1,15)
-3	10 (5,5)	9 (4,5)	10 (3, 7)	6 (1, 5)	7 (1, 6)	9 (1, 8)
-4	12 (7,5)	10 (5,5)	10 (4, 6)	9 (3, 6)	8 (1, 7)	8 (1, 7)
-5	10 (8,2)	10 (5,5)	9 (4, 5)	9 (4, 5)	8 (3, 5)	7 (1, 6)

Table 18.1 Typical error table delivered by the soft billiard (onion) algorithm. The matrix indices represent the number of errors made allowed when computing the weight vector for the – and the + class, respectively. The matrix elements contain the total number of errors made on the test set and their split into – and + class errors, respectively. The results shown are for the aspect-dependent angle sonar data. A gaussian kernel with $\sigma = 4.4$ has been used, the total number of test examples was 104.

Acknowledgments

This work has been performed during P.R.'s visit at SITE, University of Ottawa in September 1998. During the editing of this manuscript we were informed about a similar work by R. Herbrich, T. Graepel, and C. Campbell, arriving at similar conclusions [Herbrich et al., 1999a]. We thank them for sending us a reprint of their work. The kernel implementation of the billiard algorithm as described above can be found at http://www.neuro.uni-oldenburg/~rujan.

18.8 Appendix

In this section, we first present a sampling lemma (that directly follows from Hoeffding's inequality) and then discuss its implications on the number of samples needed to find a good estimate of the center of mass of the version space.

Lemma 18.1
Let $\mathbf{v}$ be a n-dimensional vector $(v_1, v_2, \ldots, v_n)$ where each component v_i is confined to the real interval $[a, b]$. Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ be k independent vectors that are identically distributed according to some unknown probability distribution P. Let μ be the true mean of $\mathbf{v}$ (*i.e.*, $\mu = \int \mathbf{v} dP(\mathbf{v})$) and let $\hat{\mu}$ be the empirical estimate of μ obtained from k samples (*i.e.*, $\hat{\mu} = (1/k) \sum_{j=1}^{k} \mathbf{v}_j$). Let $||\hat{\mu} - \mu||$ denote the Euclidean distance between $\hat{\mu}$ and μ. Then with probability at least $1-\eta$, $||\hat{\mu}-\mu|| < \epsilon$ whenever

$$k > \frac{n(b-a)^2}{2\epsilon^2} \ln\left(\frac{2n}{\eta}\right) \tag{18.29}$$

Proof To have $||\hat{\mu}-\mu|| < \epsilon$, it is sufficient to have $|\hat{\mu}_i-\mu_i| < \epsilon/\sqrt{n}$ simultaneously for each component $i = 1, \ldots, n$. Let A_i be the event for which $|\hat{\mu}_i - \mu_i| < \epsilon/\sqrt{n}$ and let $\overline{A}_i$ be the complement of that event so that $\Pr(A_1 \cap A_2 \cdots \cap A_n) = 1 - \Pr(\overline{A}_1 \cup \overline{A}_2 \cdots \cup \overline{A}_n)$. Hence we have $\Pr(A_1 \cap A_2 \cdots \cap A_n) > 1 - \eta$ if and only if $\Pr(\overline{A}_1 \cup \overline{A}_2 \cdots \cup \overline{A}_n) < \eta$. However, from the well known union bound, we have $\Pr(\overline{A}_1 \cup \overline{A}_2 \cdots \cup \overline{A}_n) \leq \sum_{i=1}^{n} \Pr(\overline{A}_i)$. Hence to have $\Pr(A_1 \cap A_2 \cdots \cap A_n) > 1-\eta$, it is sufficient to have $\Pr(\overline{A}_i) < \eta/n$ for $i = 1, \ldots, n$. Consequently, in order to have $||\hat{\mu} - \mu|| < \epsilon$ with probability at least $1 - \eta$, it is sufficient to have $|\hat{\mu}_i - \mu_i| > \epsilon/\sqrt{n}$ with probability at most η/n for each component i. Now, for any component i, Hoeffding's inequality [Hoeffding, 1963] states that:

$$\Pr\{|\hat{\mu}_i - \mu_i| \geq \alpha\} \leq 2e^{-2k\alpha^2/(b-a)^2} \tag{18.30}$$

The lemma then follows by choosing $\alpha = \epsilon/\sqrt{n}$ and by imposing that η/n be larger than the right-hand side of Hoeffding's inequality. ■

To apply this lemma to the problem of estimating the center of mass of the version space, we could just substitute for the vector $\mathbf{v}$, the separating weight vector (that would include an extra component for the threshold). However, we immediately run into a difficulty when the separating vector lies in an infinite-dimensional feature space. In that case, we just apply the lemma for the m-dimensional vector $(\alpha_1, \alpha_2, \ldots, \alpha_m)$, dual to the separating weight vector. Hence, because we must now replace n by the number m of training examples, the number k of samples needed becomes:

$$k > \frac{m(b-a)^2}{2\epsilon^2} \ln\left(\frac{2m}{\eta}\right) \tag{18.31}$$

Regardless of whether we are sampling directly the weight vectors or the duals, the bound obtained from this lemma for estimating the center of mass applies only when we are sampling according to the true Bayesian posterior distribution.

19 Margin Distribution and Soft Margin

John Shawe-Taylor
Department of Computer Science
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
j.shawe-taylor@dcs.rhbnc.ac.uk
http://www.cs.rhbnc.ac.uk/people/staff/shawe-taylor.shtml

Nello Cristianini
Department of Engineering Mathematics, University of Bristol
Queen's Building, University Walk
Bristol BS8 1TR, UK
nello.cristianini@bristol.ac.uk
http://zeus.bris.ac.uk/~ennc/nello.html

Typical bounds on generalization of Support Vector Machines are based on the minimum distance between training examples and the separating hyperplane. There has been some debate as to whether a more robust function of the margin distribution could provide generalization bounds. Freund and Schapire [1998] have shown how a different function of the margin distribution can be used to bound the number of mistakes of an on-line learning algorithm for a perceptron, as well as to give an expected error bound.

We show that a slight generalization of their construction can be used to give a pac style bound on the tail of the distribution of the generalization errors that arise from a given sample size. Furthermore, we show that the approach can be viewed as a change of kernel and that the algorithms arising from the approach are exactly those originally proposed by Cortes and Vapnik [1995]. Finally, we discuss the relations of this approach with other techniques, such as regularization and shrinkage methods.[1]

1. Parts of this work have appeared in [Shawe-Taylor and Cristianini, 1999b,a]

19.1 Introduction

The presence of noise in the data introduces a trade-off in every learning problem: complex hypotheses can be very accurate on the training set, but have worse predictive power than simpler and slightly inaccurate hypotheses. Hence the right balance between accuracy and simplicity of a hypothesis needs to be sought and this is usually attained by minimizing a cost function formed of two parts, one describing the complexity of the hypothesis, the other measuring its training error. In the case of linear functions this leads to an additional difficulty as the problem of minimizing the number of training errors is computationally infeasible if we parametrize the problem in terms of the dimension of the inputs [Arora et al., 1997]. We avoid this apparent impasse by bounding the generalization in terms of a different function of the training set performance, namely one based on the distribution of margin values, but not directly involving training error. We will show in this paper that minimizing this new criterion can be performed efficiently.

When considering large margin classifiers, where the complexity of a hypothesis is measured by its margin with respect to the data, the presence of noise can lead to further problems, for example datasets may be non-separable, and hence their margin would be negative, making application of the non-agnostic result impossible. Moreover solutions found by maximizing the margin are not stable with respect to the training points – slight modifications in the training set can significantly change the hypothesis – a brittleness which makes the maximal margin solution somehow undesirable. These problems have led to the technique of the "soft-margin," a procedure aimed at extending the large margin algorithms to the noisy case by permitting a trade-off between accuracy and margin.

non-separable data

Despite successes in extending this style of analysis to the agnostic case [Bartlett, 1998] (see (1.46) in this book) and applying it to neural networks [Bartlett, 1998], boosting [Schapire et al., 1998], and Bayesian algorithms [Cristianini et al., 1998], there has been concern that the measure of the distribution of margin values attained by the training set is largely ignored in a bound in terms of its minimal value. Intuitively, there appeared to be something lost in a bound that depended so critically on the positions of possibly a small proportion of the training set.

Though more robust algorithms have been introduced, the problem of robust bounds has remained open until recently. Freund and Schapire [1998] showed that for on-line learning a measure of the margin distribution can be used to give mistake bounds for a perceptron algorithm, and a bound on the expected error. Following a similar technique, in this paper we provide theoretical *pac* bounds on generalization using a more general function of the margin distribution achieved on the training set; we show that this technique can be viewed as a change of kernel and that algorithms arising from the approach correspond exactly to those originally proposed by Cortes and Vapnik [1995] as techniques for agnostic learning. Finally, we will show that the algorithms obtained in this way are intimately related to certain techniques, usually derived in the framework of regularization or of Bayesian

margin distribution

analysis and hence this work can be used to provide a learning-theoretic justification for such techniques.

Note that this style of analysis can also be used to transfer other hard margin results into a soft margin setting, and furthermore it can be extended to cover the nonlinear and regression cases [Shawe-Taylor and Cristianini, 1998].

19.2 Margin Distribution Bound on Generalization

We consider learning from examples of a binary classification. We denote the domain of the problem by X and a sequence of inputs by $\mathbf{x} = (x_1, \ldots, x_m) \in X^m$. A training sequence is typically denoted by $\mathbf{z} = ((x_1, y_1), \ldots, (x_m, y_m)) \in (X \times \{-1, 1\})^m$ and the set of training examples by S. By $\mathrm{Er}_z(f)$ we denote the number of classification errors of the function f on the sequence $\mathbf{z}$.

As we will typically be classifying by thresholding real valued functions we introduce the notation $T_\theta(f)$ to denote the function giving output 1 if f has output greater than or equal to θ and -1 otherwise. For a class of real-valued functions $\mathcal{H}$ the class $T_\theta(\mathcal{H})$ is the set of derived classification functions.

fat shattering dimension

Definition 19.1
Let F be a set of real valued functions. We say that a set of points X is γ-*shattered by* F if there are real numbers r_x indexed by $x \in X$ such that for all binary vectors b indexed by X, there is a function $f_b \in F$ satisfying $f_b(x) \geq r_x + \gamma$, if $b_x = 1$ and $f_b(x) \leq r_x - \gamma$, otherwise.

The relevance of the fat shattering dimension and margin for learning is illustrated in the following theorem which bounds the generalization error in terms of the fat shattering dimension of the underlying function class measured at a scale proportional to the margin.

Theorem 19.2 Shawe-Taylor, Bartlett, Williamson, and Anthony, 1998

Consider a real valued function class F having fat-shattering dimension bounded above by the function $\mathrm{fat} : \mathbb{R} \to \mathbb{N}$ which is continuous from the right. Fix $\theta \in \mathbb{R}$. Then with probability at least $1 - \delta$ a learner who correctly classifies m independently generated examples S with $h = T_\theta(f) \in T_\theta(F)$ such that $\gamma = \min_i y_i(f(x_i) - \theta) > 0$ will have the error of h bounded from above by

$$\epsilon(m, k, \delta) = \frac{2}{m}\left(k \log_2\left(\frac{8em}{k}\right)\log_2(32m) + \log_2\left(\frac{8m}{\delta}\right)\right),$$

where $k = \mathrm{fat}(\gamma/8) \leq em$.

The first bound on the fat shattering dimension of bounded linear functions in a finite dimensional space was obtained by Shawe-Taylor et al. [1998]. Gurvits [1997] generalized this to infinite dimensional Banach spaces (see Theorem 1.12 for an improved version thereof).

We first summarize results from [Shawe-Taylor and Cristianini, 1999b]. Let X be an inner product space. We define the following inner product space derived from X.

Definition 19.3
Let $L_f(X)$ be the set of real valued functions f on X with countable support $\text{supp}(f)$ (that is functions in $L_f(X)$ are non-zero for only countably many points) for which the sum of the squared values

$$\|f\|^2 = \sum_{x \in \text{supp}(f)} f(x)^2$$

converges. We define the inner product of two functions $f, g \in L_f(X)$, by

$$\langle f, g \rangle = \sum_{x \in \text{supp}(f)} f(x)g(x).$$

Note that the sum which defines the inner product can be shown to converge by using the Cauchy-Schwartz inequality on the difference of partial sums and hence showing that the partial sums form a Cauchy sequence. Clearly the space is closed under addition and multiplication by scalars.

map to a separation space

Now for any fixed $\Delta > 0$ we define an embedding of X into the inner product space $X \times L_f(X)$ as follows: $\tau_\Delta : x \mapsto (x, \Delta\delta_x)$, where $\delta_x \in L_f(X)$ is defined by $\delta_x(y) = 1$, if $y = x$ and 0, otherwise. Embedding the input space X into $X \times L_f(X)$ maps the training data into a space where it can be separated by a large margin classifier and hence we can apply Theorem 19.2. The cost of performing this separation appears in the norm of the linear operator acting in $L_f(X)$ which forces the required margin. The following definition specifies the amount by which a training point has to be adjusted to reach the desired margin γ.

For a linear classifier $(\mathbf{u}, b)$ on X and margin $\gamma \in \mathbb{R}$ we define

$$d((x, y), (\mathbf{u}, b), \gamma) = \max\{0, \gamma - y(\langle \mathbf{u}, x \rangle - b)\}.$$

This quantity is the amount by which $(\mathbf{u}, b)$ fails to reach the margin γ on the point (x, y) or 0 if its margin is larger than γ. For a misclassified point (x, y) we will have $d((x, y), (\mathbf{u}, b), \gamma) > \gamma$, and so misclassification is viewed as a worse margin error, but is not distinguished into a separate category. We now augment $(\mathbf{u}, b)$ to the linear functional

$$\hat{\mathbf{u}} = \left(\mathbf{u}, \frac{1}{\Delta} \sum_{(x,y) \in S} d((x, y), (\mathbf{u}, b), \gamma) y \delta_x \right).$$

in the space $X \times L_f(X)$. The action of the additional component is exactly enough to ensure that those training points that failed to reach margin γ in the input space now do so in the augmented space. The cost of the additional component is in its effect of increasing the square of the norm of the linear functional by

$D(S,(\mathbf{u},b),\gamma)^2/\Delta^2$, where

$$D(S,(\mathbf{u},b),\gamma) = \sqrt{\sum_{(x,y)\in S} d((x,y),(\mathbf{u},b),\gamma)^2}. \tag{19.1}$$

At the same time the norm of the training points has been increased by the additional component $\Delta\delta_x$. Taking both these adjustments into account and verifying that the off-training set performance of the augmented classifier matches exactly the original linear function gives the following theorem as a consequence of Theorems 19.2 and 1.12.

bound for a fixed map

Theorem 19.4 Shawe-Taylor and Cristianini [1999b]
Fix $\Delta > 0$, $b \in \mathbb{R}$. Consider a fixed but unknown probability distribution on the input space X with support in the ball of radius R about the origin. Then with probability $1-\delta$ over randomly drawn training sets S of size m for all $\gamma > 0$ the generalization of a linear classifier $\mathbf{u}$ on X with $\|\mathbf{u}\| = 1$, thresholded at b is bounded by

$$\epsilon(m,h,\delta) = \frac{2}{m}\left(h\log_2\left(\frac{8em}{h}\right)\log_2(32m) + \log_2\left(\frac{8m}{\delta}\right)\right),$$

where

$$h = \left\lfloor \frac{64.5(R^2+\Delta^2)(1+D(S,(\mathbf{u},b),\gamma)^2/\Delta^2)}{\gamma^2} \right\rfloor,$$

provided $m \geq 2/\epsilon$, $h \leq em$ and there is no discrete probability on misclassified training points.

Note that unlike Theorem 19.2 the theorem does not require that the linear classifier $(\mathbf{u},b)$ correctly classifies the training data. Misclassified points will contribute more to the quantity $D(S,(\mathbf{u},b),\gamma)$, but will not change the structure of the result. This contrasts with their effect on Theorem 19.2 where resorting to the agnostic version introduces a square root into the expression for the generalization error.

In practice we wish to choose the parameter Δ in response to the data in order to minimize the resulting bound. In order to obtain a bound which holds for different values of Δ it will be necessary to apply the Theorem 19.4 several times for a finite subset of values. Note that the minimum of the expression for h (ignoring the constant and suppressing the denominator γ^2) is $(R+D)^2$ attained when $\Delta = \sqrt{RD}$. The discrete set of values must be chosen to ensure that we can get a good approximation to this optimal value. The solution is to choose a geometric sequence of values – see [Shawe-Taylor and Cristianini, 1999b] for details.

bound for optimal map

Theorem 19.5 Shawe-Taylor and Cristianini [1999b]
Fix $b \in \mathbb{R}$. Consider a fixed but unknown probability distribution on the input space X with support in the ball of radius R about the origin. Then with probability $1-\delta$ over randomly drawn training sets S of size m for all $\gamma > 0$ such that $d((x,y),(\mathbf{u},b),\gamma) = 0$, for some $(x,y) \in S$, the generalization of a linear classifier $\mathbf{u}$

on X satisfying $\|\mathbf{u}\| \leq 1$ is bounded by

$$\epsilon(m,h,\delta) = \frac{2}{m}\left(h\log_2\left(\frac{8em}{h}\right)\log_2(32m) + \log_2\left(\frac{2m(28+\log_2(m))}{\delta}\right)\right),$$

where

$$h = \left\lfloor \frac{65[(R+D)^2 + 2.25RD]}{\gamma^2} \right\rfloor,$$

for $D = D(S,(\mathbf{u},b),\gamma)$, and provided $m \geq \max\{2/\epsilon, 6\}$, $h \leq em$ and there is no discrete probability on misclassified training points.

As discussed above the bound can be used for classifiers that misclassify some training points. The effect of misclassified points will only be felt in the value of D. Such points do not change the form of the expression. This is in contrast with traditional agnostic bounds which involve the square root of the ratio of the fat shattering dimension and sample size (see for example expression (1.46) in this book). If a point is an extreme outlier, it is possible that its effect on D might be such that the bound will be worse than that obtained using the agnostic approach (where the "size" of misclassification is irrelevant). However, it is likely that in usual situations the bound given here will be significantly tighter than the standard agnostic one. The other advantage of the new bound will be discussed in the next section where we show that in contrast to the computational difficulty of minimizing the number of misclassifications, there exists an efficient algorithm for optimizing the value of h given in Theorem 19.5.

19.3 An Explanation for the Soft Margin Algorithm

The theory developed in the previous section provides a way to transform a non linearly separable problem into a separable one by mapping the data to a higher dimensional space, a technique that can be viewed as using a kernel in a similar way to Support Vector Machines.

Is it possible to give an effective algorithm for learning a large margin hyperplane in this augmented space? This would automatically give an algorithm for choosing the hyperplane and value of γ, which result in a margin distribution in the original space for which the bound of Theorem 19.5 is minimal. It turns out that not only is the answer yes, but also that such an algorithm already exists.

separation kernels

The mapping τ defined in the previous section implicitly defines a kernel as follows:

$$\begin{aligned} k(x,x') &= \langle \tau_\Delta(x), \tau_\Delta(x') \rangle \\ &= \langle (x, \Delta\delta_x), (x', \Delta\delta_{x'}) \rangle \\ &= \langle x, x' \rangle + \Delta^2 \langle \delta_x, \delta_{x'} \rangle \\ &= \langle x, x' \rangle + \Delta^2 \delta_x(x') \end{aligned}$$

By using these kernels, the decision function of a SV machine would be:

$$f(x) = \sum_{i=1}^{m} \alpha_i y_i k(x, x_i) + b$$
$$= \sum_{i=1}^{m} \alpha_i y_i \left[\langle x, x_i \rangle + \Delta^2 \delta_x(x_i) \right] + b$$

and the Lagrange multipliers α_i would be obtained by solving the Quadratic Programming problem of minimizing in the positive quadrant the dual objective function:

$$L = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j k(x_i, x_j)$$
$$= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j [\langle x_i, x_j \rangle + \Delta^2 \delta_i(j)]$$
$$= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle - \Delta^2 \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \delta_i(j)$$
$$= \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle - \Delta^2 \frac{1}{2} \sum_{i=1}^{m} \alpha_i^2$$

This is exacly the dual QP problem that one would obtain by solving the soft margin problem in one of the cases stated in the appendix of [Cortes and Vapnik, 1995]:

soft margin

$$\text{minimize:} \quad \frac{1}{2} \langle \mathbf{u}, \mathbf{u} \rangle + C \sum \xi_i^2$$

$$\text{subject to:} \quad y_j [\langle \mathbf{u}, x_j \rangle - b] \geq 1 - \xi_j$$
$$\xi_i \geq 0$$

The solution they obtain is:

$$L = \sum \alpha_i - \sum y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle - \frac{1}{4C} \sum \alpha_i^2$$

which makes clear how the trade off parameter C in their formulation is related to the kernel parameter Δ.

19.4 Related Techniques

Another way of looking at this technique is that optimizing the soft margin, or enlarging the margin distribution, is equivalent to replacing the covariance matrix K with the covariance K'

covariance of augmented data

$$K' = K + \lambda I$$

which has a heavier diagonal. Again, there is a simple relationship between the trade off parameter λ and the Δ and C of the previous formulations. So rather than using a soft margin algorithm, one can use a (simpler) hard margin algorithm after adding λI to the covariance matrix. This approach has also been considered by Smola and Schölkopf [1998b] for the regression case where they also introduce an upper bound on the size of the α's in order to improve robustness to outliers.

Figure 19.4 shows the results of experiments performed on the ionosphere data of the UCI repository [Blake et al., 1998]. The plot is of the generalization error for different values of the parameter λ.

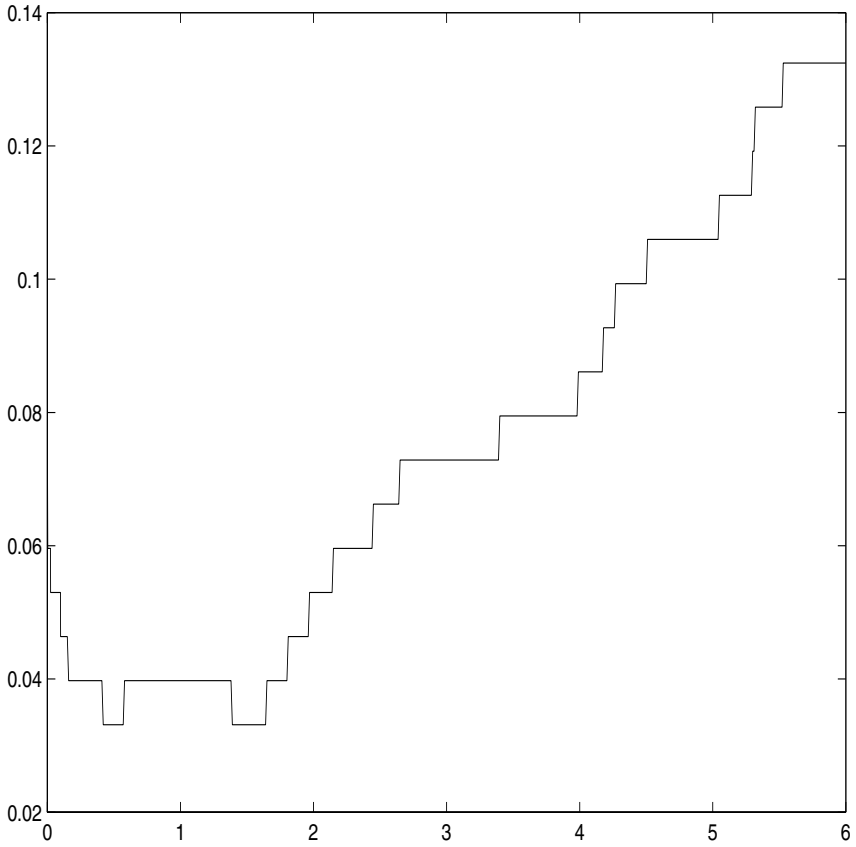


Figure 19.1 Generalization error as a function of λ, in a hard margin problem with augmented covariance $K' = K + \lambda I$, for *ionosphere* data.

equivalent techniques

This technique is well known in classical statistics, where it is sometimes called the "shrinkage method" (see Ripley [1996]). Basically, in Bayesian discrimination (see Section 1.1.1) it suggests replacing the empirical covariance function Σ with some

function closer to the identity I, by choosing an element of the line joining them $(1-\lambda)\Sigma + \lambda I$. A redundant degree of freedom is then removed, leaving with the new covariance $\Sigma + \lambda I$. In the case of linear regression this technique, known as ridge regression, can be derived from assuming Gaussian noise on the target values. It was originally motivated by the trade off between bias and variance [Hoerl and Kennard, 1970] and leads to a form of weight decay. This approach is equivalent to a form of regularization in the sense of Tikhonov. The theory of ill-posed problems was developed by Tikhonov in the context of solving inverse problems [Tikhonov and Arsenin, 1977]. Smola and Schölkopf [1998b] derived ridge regression using dual variables and for example Vovk et al. [1998] have applied this to benchmark problems. It is well known that one can perform regularization by replacing the covariance matrix X^TX with $X^TX + \lambda I$, and learning machines based on Gaussian Processes implicitly exploit this fact in addition to the choice of kernel.

Another explanation proposed for the same technique is that it reduces the number of effective free parameters, as measured by the trace of K. Note finally that from an algorithmical point of view these kernels still give a positive definite matrix, and a better conditioned problem than the hard margin case, since the eigenvalues are all increased by λ. The so-called box constraint algorithm which minimizes the 1-norm of the slack variables is not directly comparable with the 2-norm case considered here.

Remark 19.6
Note that

$$R\sqrt{\sum \xi_i^2} = RD = \Delta^2 = \lambda = \frac{1}{4C}$$

so a choice of γ in the margin distribution bound controls the parameter C in the soft margin setting, and the trade-off parameter λ in the regularization setting. A reasonable choice of γ can be one that minimizes some VC bound on the capacity, for example maximizing the margin in the augmented space, or controlling other parameters (margin; eigenvalues; radius; etc). Note also that this formulation also makes intuitive sense: a small γ corresponds to a small λ and to a large C: little noise is assumed, and so there is little need for regularization; vice versa a large γ corresponds to a large λ and a small C, which corresponds to assuming a high level of noise. Similar reasoning leads to similar relations in the regression case.

19.5 Conclusion

The analysis we have presented provides a principled way to deal with noisy data in large margin classifiers, and justifies the like the soft margin algorithm as originally proposed by Cortes and Vapnik. We have proved that one such algorithm exactly minimizes the bound on generalization provided by our margin distribution analysis, and is equivalent to using an augmented version of the kernel. Many techniques developed for the hard margin case can then be extended to the soft-margin case,

as long as the quantities they use can be measured in terms of the modified kernel (margin, radius of the ball, eigenvalues).

The algorithms obtained in this way are strongly related to regularization techniques, and other methods developed in different frameworks in order to deal with noise. Computationally, the algorithm can be more stable and better conditioned than the standard maximal margin approach.

Finally, the same proof technique can also be used to produce analogous bounds for nonlinear functions in the classification case, and for the linear and nonlinear regression case with different losses, as reported in the full paper [Shawe-Taylor and Cristianini, 1998].

Acknowledgments

This work was supported by the European Commission under the Working Group Nr. 27150 (NeuroCOLT2) and by the UK EPSRC funding council. The authors would like to thank Colin Campbell and Bernhard Schölkopf for useful discussions. They would also like to thank useful comments from an anonymous referee that helped to refine Definition 19.3.

20 Support Vectors and Statistical Mechanics

Rainer Dietrich
Institut für Theoretische Physik
Julius-Maximilians-Universität
Am Hubland
D-97074 Würzburg, Germany
dietrich@physik.uni-wuerzburg.de
http://theorie.physik.uni-wuerzburg.de/~dietrich

Manfred Opper
Department of Computer Science and Applied Mathematics
Aston University
Aston Triangle
Birmingham B4 7ET, UK
opperm@aston.ac.uk
http://www.ncrg.aston.ac.uk/People/opperm/Welcome.html

Haim Sompolinsky
Racah Institute of Physics and Center for Neural Computation
Hebrew University
Jerusalem 91904, Israel
haim@fiz.huji.ac.il

We apply methods of Statistical Mechanics to study the generalization performance of Support Vector Machines in large dataspaces.

20.1 Introduction

Many theoretical approaches for estimating the generalization ability of learning machines are based on general, distribution independent bounds. Since such bounds hold even for very unfavourable data generating mechanisms, it is not clear a priori how tight they are in less pessimistic cases.

Hence, it is important to study models of nontrivial learning problems for which we can get exact results for generalization errors and other properties of a trained learning machine. A method for constructing and analysing such learning situations has been provided by Statistical Mechanics. Statistical Mechanics is a field of Theoretical Physics which deals with a probabilistic description of complex systems that are composed of many interacting entities. Tools originally developped to study the properties of amorphous materials enable us to conduct controlled, *analytical* experiments for the performance of learning machines for specific types of data distributions when the numbers of tunable parameters and examples are large. While often statistical theories provide asymptotic results for sizes of the training data sample that are much larger than some intrinsic complexity of a learning machine, in contrast, the so called 'thermodynamic limit' of Statistical Mechanics allows to simulate the effects of small *relative* sample sizes. This is achieved by taking the limit where both the sample size and the number of parameters approaches infinity, but an appropriate ratio is kept fixed.

Starting with the pioneering work of Elizabeth Gardner [1988] this approach has been successfully applied during the last decade to a variety of problems in the context of neural networks (for a review, see, e.g., [Seung et al., 1992, Watkin et al., 1993, Opper and Kinzel, 1996]). This chapter will deal with an application to learning with Support Vector Machines (SVMs). A somewhat more detailed analysis which was designed for readers with a Statistical Physics background, can be found in [Dietrich et al., 1999].

20.2 The Basic SVM Setting

We will restrict ourselves to SVM classifiers. They are defined (for more explanations, see the introductory chapter to this book) by a nonlinear mapping $\Phi(\cdot)$ from input vectors $\mathbf{x} \in \mathbb{R}^N$ into a feature space $\mathcal{F}$. The mapping is constructed from the eigenvectors $\psi_j(\mathbf{x})$ and eigenvalues λ_j of an SVM kernel $k(\mathbf{x}, \mathbf{y})$ via $\Phi(\mathbf{x}) = (\sqrt{\lambda_1}\psi_1(\mathbf{x}), \sqrt{\lambda_2}\psi_2(\mathbf{x}), \ldots)$.

The output y of the SVM can be represented as a linear classification

$$\text{sgn}\ (\Phi(\mathbf{x}) \cdot \mathbf{w}) = \text{sgn}\ \left(\sum_{j=1}^{N_\mathcal{F}} \sqrt{\lambda_j}\psi_j(\mathbf{x}) w_j \right) \tag{20.1}$$

in feature space, where for simplicity, we have set the bias term equal to zero. For a realizable setting, the weights w_j, $j = 1, \ldots, N_\mathcal{F}$ are adjusted to a set of example pairs $\{(y_1, \mathbf{x}_1), \ldots, (y_m, \mathbf{x}_m)\}$ by minimizing the quadratic function $\frac{1}{2}||\mathbf{w}||^2$ under the constraints that $y(\Phi(\mathbf{x}) \cdot \mathbf{w}) \geq 1$ for all examples.

20.3 The Learning Problem

teacher-student framework

We assume a simple noise free scenario, where the generation of data is modelled within the so called teacher-student framework. Here, it is assumed that some classifier (the teacher) which has a similar representation as the machine of interest, gives the correct outputs to a set of randomly generated input data. The generalization error can be measured as the probability of disagreement on a random input between teacher and student machine. In our case, we choose the representation

$$y_i = \operatorname{sgn}\left(\sum_j \sqrt{\lambda_j} B_j \psi_j(\mathbf{x}_i)\right). \tag{20.2}$$

All nonzero components are assumed to be chosen independently at random from a distribution with zero mean and unit variance. We will also consider the case, where a finite fraction of the B_j are 0 in order to tune the complexity of the rule. Finally, the inputs $\mathbf{x}_i$ are taken as independent random vectors with a uniform probability distribution $D(\mathbf{x})$ on the hypercube $\{-1,1\}^N$. We are interested in the performance of the SVM averaged over these distributions.

We will specialize on a family of kernels which have the form $k(\mathbf{x},\mathbf{y}) = K\left(\frac{\mathbf{x}\cdot\mathbf{y}}{N}\right)$, where, for simplicity, we set $K(0) = 0$. These kernels are permutation symmetric in the components of the input vectors and contain the simple perceptron margin classifier as a special case, when $K(z) = z$. For binary input vectors $\mathbf{x} \in \{-1,1\}^N$, the eigenvalue decomposition for this type of kernels is known [Kühn and van Hemmen, 1996]. The eigenfunctions are products of components of the input vectors, i.e., $\psi_i(\mathbf{x}) = 2^{-N/2}\prod_{j\in S_i} x_j$, which are simple monomials, where $S_i \subseteq \{1,\ldots,N\}$ is a subset of the components of $\mathbf{x}$. For polynomial kernels, these features have also been derived in [Smola et al., 1998a]. The corresponding eigenvalues are found to be $\lambda_i = 2^{N/2}\sum_{\mathbf{x}} k(\mathbf{e},\mathbf{x})\psi_i(\mathbf{x})$, with $\mathbf{e} = (1,\ldots,1)^T$. They depend on the cardinality $|S_i|$ of the set S_i only. For $|S_i| = 1$, the eigenfunctions are the N linear functions x_j, $j = 1,\ldots,N$. For $|S_i| = 2$, we have the $N(N-1)/2$ bilinear combinations $x_i x_j$ etc. The behaviour of the eigenvalues for large input dimension N is given by $\lambda_i \simeq \frac{2^N}{N^{|S_i|}} K^{(|S_i|)}(0)$. $K^{(l)}$ denotes the l-th derivative of the function K. The rapid decrease of the eigenvalues with the cardinality $|S_i|$ is counterbalanced by the strong increase of their degeneracy which grows like $n_{|S_i|} = \binom{N}{|S_i|} \simeq N^{|S_i|}/|S_i|!$. This keeps the overall contribution of eigenvalues $\sum_{|S_i|=l} \lambda_i n_{|S_i|}$ for different cardinalities l of the same order.

eigenvalue decomposition

20.4 The Approach of Statistical Mechanics

The basic idea to map SVM learning to a problem in Statistical Mechanics is to define a (Gibbs) measure $p_\beta(\mathbf{w})$ over the weights $\mathbf{w}$ which in a specific limit is

concentrated at the weights of the trained SVM. This is done by setting

$$p_\beta(\mathbf{w}) = \frac{1}{Z} e^{-\frac{1}{2}\beta||\mathbf{w}||^2} \prod_{i=1}^{m} \Theta\left(y_i \sum_{j=1}^{N_\mathcal{F}} \sqrt{\lambda_j}\psi_j(\mathbf{x}_i) w_j - 1 \right). \tag{20.3}$$

$\Theta(x)$ is the unit step function which equals 1 for $x \geq 0$ and 0 else. Z normalizes the distribution. In the limit $\beta \to \infty$, this distribution is concentrated at the minimum of $||\mathbf{w}||^2$ in the subspace of weights where all arguments of the Θ functions are nonnegative. This is equivalent to the conditions of the SVM quadratic programming problem. A different approach has been discussed in [Opper, 1999], where the Kuhn Tucker conditions of the optimization problem have been directly implemented into a Statistical Mechanics framework. It will be interesting to see, if this method can also be applied to the generalization problem of SVMs.

The strategy of the Statistical Mechanics approach consists of calculating expectations of interesting quantities which are functions of the weight vector $\mathbf{w}$ over both the distribution (20.3) and over the distribution of the training data. At the end of the calculation, the limit $\beta \to \infty$ is taken. These averaging procedures can be performed analytically only in the limit where $N \to \infty$ and $m \to \infty$. They require a variety of delicate and nontrivial manipulations which for lack of space cannot be explained in this contribution. One of these techniques is to apply a central limit theorem (valid in the 'thermodynamic limit') for carrying out expectations over the random inputs, utilizing the fact that the features ψ_j are orthogonal with respect to the chosen input distribution. This is the main reason, why we prefer to work in high-dimensional feature space rather than using the low dimensional kernel representation. A review of the standard techniques used in the Statistical Mechanics approach and their application to the generalization performance of neural networks can be found, e.g., in [Seung et al., 1992, Watkin et al., 1993, Opper and Kinzel, 1996]), a general review of the basic principles is [Mézard et al., 1987].

thermodynamic limit

The results of our analysis will depend on the way, in which the two limits $N \to \infty$ and $m \to \infty$ are carried out. In general, one expects that a decay of the generalization error ϵ_g to zero should occur only when $m = \mathcal{O}(N_\mathcal{F})$, because $N_\mathcal{F}$ is the number of parameters of the data model. Nevertheless, when the mapping Φ contains a reasonably strong linear part, ϵ_g may drop to small values already on a scale of $m = \alpha N$ examples. Hence, in taking the limit $N \to \infty$, we will make the general ansatz $m = \alpha N^l$, $l \in \mathbb{N}$ and discuss different regions of the generalization performance by varying l. Our model differs from a previous Statistical Mechanics approach to SVMs [Buhot and Gordon, 1999] where the dimension of the feature space grew only linear with N.

20.5 Results I: General

One of the most basic and natural quantities which result from the calculation is a so called order parameter which for the SVM is defined by

$$R = \sum_i \Lambda_i \langle w_i B_i \rangle \tag{20.4}$$

where $\Lambda_i := \lambda_i / 2^N$, and $\langle ... \rangle$ denotes an average with respect to the distribution (20.3) and the distributions of the data and of the teacher vector. R is a weighted overlap between the teacher and SVM student weight vectors. This similarity measure between teacher and student allows us to express the generalization error by $\epsilon_g = \frac{1}{\pi} \arccos \frac{R}{\sqrt{Bq}}$. Here $B = \sum_i \Lambda_i \langle (B_i)^2 \rangle$ and $q_0 = \sum_i \Lambda_i \langle (w_i)^2 \rangle$ denote specific squared norms of the teacher and student weight vectors. Note that by the specific form of ϵ_g, the teacher's rule is perfectly learnt when the student vector points in the same direction as the teacher irrespectively of the student vector's length. Furthermore, an analysis of the contributions coming from eigenvectors of different complexities (i.e., cardinalities $|S_i|$) will give us an intuitive understanding of the SVMs inference of the rule.

generalization error

As a general result of our analysis, we find that if the number of examples is scaled as $m = \alpha N^l$,

scaling of number of inputs

- All high order components B_i are completely undetermined, i.e., $R^{(+)} := \sum_{|S_i|>l} \Lambda_i \langle w_i B_i \rangle \to 0$, and also that $q_0^{(+)} := \sum_{|S_i|>l} \Lambda_i \langle (w_i)^2 \rangle \to 0$, in the large N limit.

This does not mean that the values of the corresponding weights w_i are zero, they are just too small to contribute in the limit to the weighted sums (20.4).

- All low order components are *completely* determined, in the sense that $w_i = cB_i$ for all i with $|S_i| < l$, where c depends on α only. The only components which are actually learnt at a scale l are those for $|S_i| = l$.

To illustrate this behaviour for the simplest case, we study quadratic kernels of the form $K(x) = (1-d)x^2 + dx$, where the parameter d, $0 < d < 1$, controls the nonlinearity of the SVM's mapping. The eigenvectors of lowest complexity are just the N linear monomials $\sim x_j$, and the remaining ones are the $N(N-1)/2$ quadratic terms of the form $x_i x_j$. The learning curve is shown in Figure 20.1, where we have included results from simulations for comparison.

If the number of examples scales linearly with the input dimension, i.e., $m = \alpha N$ (left side of Figure 20.1), the SVM is able to learn only the linear part of the teacher's rule. However, since there is not enough information to infer the remaining $N(N-1)/2$ weights of the teacher's quadratic part, the generalization error of the SVM reaches a nonzero plateau as $\alpha \to \infty$ according to $\epsilon_g(\alpha) - \epsilon_g(\infty) \sim \alpha^{-1}$. The height of the plateau is given by $\epsilon_g(\infty) = \pi^{-1} \arccos(d)$, which increases from zero at $d = 1$, when the kernel is entirely linear, to $\epsilon_g = \frac{1}{2}$ at $d = 0$ when only quadratic features are present.

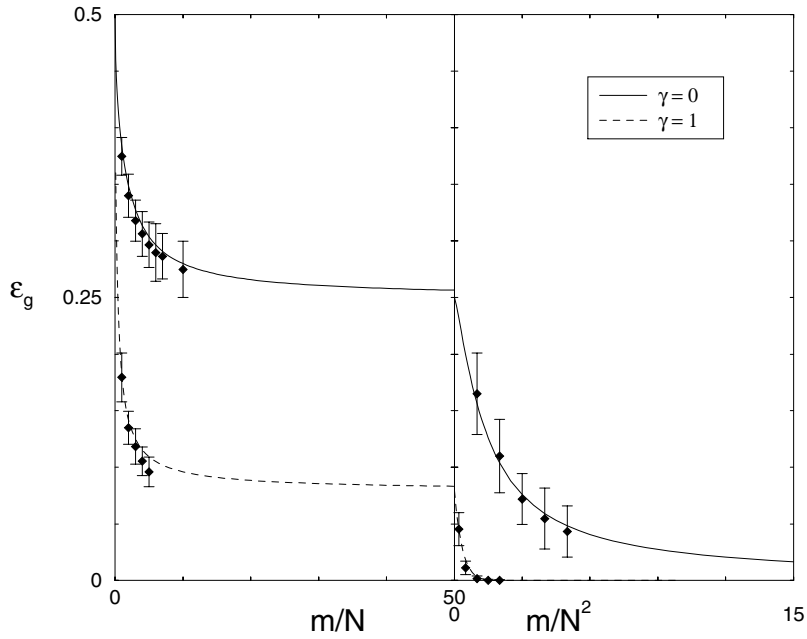


Figure 20.1 Decrease of the generalization error on different scales of examples, for quadratic SVM kernel learning a quadratic teacher rule ($d = 0.5$, $B = 1$) and various gaps γ. Simulations were performed with N= 201 and averaged over 50 runs (left and next figure), and N= 20, 40 runs (right).

If we increase the number of examples to grow quadratically with N, i.e., $m = \alpha N^2$ (right side of Figure 20.1), the generalization error will decrease towards zero with a behavior $\sim 1/\alpha$ asymptotically, where the prefactor does not depend on d.

polynomial kernels

The retarded learning of the more complex components of the mapping Φ generalizes to kernels which are polynomials of higher order $z > 2$. On the scale of $m = \alpha N^l$ examples, when $l < z$, the generalization error decreases to a plateau as $\alpha \to \infty$ which is given by

$$\epsilon_g = \frac{1}{\pi} \arccos \sqrt{\frac{\sum_{j=1}^{l} \frac{K^{(j)}(0)}{j!}}{K(1)}}. \tag{20.5}$$

Only at the highest scale $m = \alpha N^z$, we get an asymptotical vanishing of the generalization error to zero as $\epsilon_g \approx \frac{0.500489}{z!} \alpha^{-1}$.

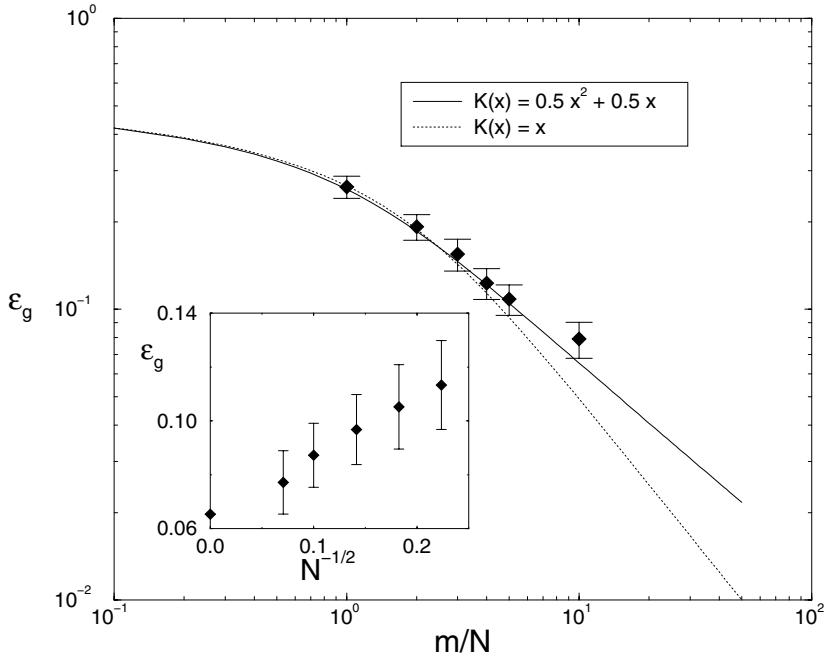


Figure 20.2 Learning curves for linear student and quadratic SVM kernels, all learning a linear teacher rule ($B = d$). For $\alpha = 10$, a finite size scaling is shown in the inset.

20.6 Results II: Overfitting

As the next problem, we study the ability of the SVM to cope with the problem of overfitting when learning a rule which has a much lower complexity than the mapping Φ. We model such a problem by keeping the SVM quadratic, but choosing a data generating mechanism which is defined by a simple *linear* separation of examples. This is achieved by setting $|B_i| = 1$ for $|S_i| = 1$ and $|B_i| = 0$ for the higher order components. Our results for the generalization error are shown in Figure 20.2, where the number of examples is scaled as $m = \alpha N$. Surprisingly, although the complexity of the SVM is by far higher than the underlying rule, only a rather weak form of overfitting is observed. The SVM is able to learn the N teacher weights B_i on the correct scale of $m = \alpha N$ examples. The asymptotic rate of convergence is $\epsilon_g \sim \alpha^{-2/3}$. If we had used a simple linear SVM for the same task, we would have learned the underlying concept only slightly faster at $\epsilon_g \sim \alpha^{-1}$.

We can compare these results with simple bounds on the expected generalization error as described in Section 1.3.4 of the introductory chapter. E.g., the expectation of the ratio of the number of support vectors over the total number of examples m yields an upper bound on ϵ_g [Vapnik, 1995]. Calculating the expected number of support vectors within the Statistical Mechanics approach yields an asymptotic decay $\sim \alpha^{-1/3}$ for this bound which decays at a slower rate than the actual ϵ_g.

20.7 Results III: Dependence on the Input Density

One can expect that if the density of inputs acts in a favourable way together with the teacher's concept, learning of the rule will be faster. We have modelled such a situation by constructing an input distribution which is *correlated* with the teacher weights B_i by having a gap of zero density of size 2γ around the teacher's decision boundary. In this case we expect to have a large margin between positive and negative examples. The density for this model is of the form $D(\mathbf{x}) \sim \Theta\left(|\sum_i \sqrt{\lambda_i} B_\rho \psi_i(\mathbf{x})| - \gamma\right)$.

For a quadratic SVM learning from a quadratic teacher rule, we observe a faster decay of the generalization error than in the case of a uniform density. However, on the linear scale $m = \alpha N$ (Figure 20.1) the asymptotic decay is still of the form $\epsilon_g(\alpha) - \epsilon_g(\infty) \sim \alpha^{-1}$. A dramatic improvement is obtained on the highest scale $m = \alpha N^2$, where the generalization error drops to zero like $\epsilon_g \sim \alpha^{-3} e^{-\hat{c}(\gamma)\alpha^2}$. In this case, the mismatch between the true generalization error and the simple bound based on the fraction of support vectors is much more striking. The latter decreases much slower, i.e., only algebraically with α.

20.8 Discussion and Outlook

The present work analysed the performance of SV Machines by methods of Statistical Mechanics. These methods give distribution dependent results on generalization errors for certain simple distributions in the limit of high dimensional input spaces.

Why do we expect that this somewhat limited approach may be of interest to the machine learning community? Some of the phenomena discussed in this chapter could definitely be observed qualitatively in other, more general approaches which are based on rigorous bounds. E.g., the recently introduced concept of *luckiness* [Shawe-Taylor et al., 1998, Schölkopf et al., 1999] applied to the case of the favourable density with a gap would give smaller generalization errors than for a uniform density. This is because the margin (taken as a luckiness function) would come out typically larger. Nevertheless, the *quantitative* agreement with the true learning curves is usually less good. Hence, an application of the bounds to model selection may in some cases lead to suboptimal results.

On the other hand, the power of the Statistical Mechanics approach comes from the fact that (in the so far limited situations, where it can be applied) it yields *quantitatively exact* results in the thermodynamic limit, with excellent agreement with the simulations of large systems. Hence, this approach can be used to check the tightness of bounds in controlled analytical experiments. We hope that it will also give an idea how bounds could be improved or replaced by good heuristics.

So far, we have restricted our results to a noise free scenario, but it is straightforward to extend the approach to noisy data. It is also possible to include SVM training with errors (resulting in the more advanced optimization problem with

slack variables) in the formalism. We expect that our analysis will give insight into the performance of model selection criteria which are used in order to tune the parameters of the SVM learning algorithm to the noise. We have already shown for the noise free case that a very simple statistics like the relative number of support vectors can give a wrong prediction for the rate of convergence of the generalization error. It will be interesting to see if more sophisticated estimates based on the margin will give tighter bounds.

Acknowledgements

This work was supported by a grant (Op 45/5-2) of the Deutsche Forschungsgemeinschaft and by the British-German Academic Research Collaboration Programme project 1037 of the British council. The work of HS was supported in part by the USA-Israel Binational Science Foundation.

21 Entropy Numbers for Convex Combinations and MLPs

Alexander J. Smola
Department of Engineering
Australian National University
Canberra 0200 ACT
Australia
Alex.Smola@anu.edu.au
http://spigot.anu.edu.au/∼smola/

André Elisseeff
Université Lyon 2,
Laboratoire ERIC,
69676 Bron Cedex
France
aelissee@univ-lyon2.fr
http://eric.univ-lyon2.fr/pages_defaut/aelissee.html

Bernhard Schölkopf
Microsoft Research Limited
St. George House, 1 Guildhall Street
Cambridge CB2 3NH
UK
bsc@microsoft.com
http://www.research.microsoft.com/∼bsc/

Robert C. Williamson
Department of Engineering
Australian National University,
Canberra, ACT, 0200
Australia
bob.williamson@anu.edu.au
http://spigot.anu.edu.au/∼williams/home.shtml

Bounds on the generalization performance of algorithms such as boosting, linear programming machines and (multilayer) RBF-networks require a good estimate of the covering or entropy numbers for the corresponding hypothesis classes. The classes are generated by convex combinations and concatenations of basis functions for which we provide functional analytic bounds on the entropy numbers.

The results are novel in three regards. First, bounds are given for vector valued functions directly without having to use a generalization of the VC dimension or other combinatorial quantities. Secondly, bounds are derived for convex combinations of parametric families. It is shown that significantly better bounds can be obtained depending on the eigenvalues of the corresponding integral operators when one deals with kernel functions. Finally, a concatenation theorem allows the use of the previously established results to images of nonlinear operators, such as the outputs of multilayer networks.

21.1 Introduction

Theoretical bounds on the generalization performance of Support Vector (SV) Machines follow from general results of statistical learning theory along with good bounds on covering numbers for the class of functions induced by such machines (cf., e.g., Section 1.2.3). Williamson et al. [1998] show how bounds can be obtained using the machinery of entropy numbers of operators. This is possible because the class of functions is defined via a restriction of a weight vector w to lie within a certain ball in feature space.

The present chapter extends and modifies the methods of [Williamson et al., 1998] in order to deal with other types of learning machines as well. These include convex combinations of hypotheses as used in boosting [Schapire et al., 1998] and linear programming machines [Bennett, 1999, Weston et al., 1999], or concatenations of hypothesis classes such as multilayer rbf networks. In particular our results apply to the algorithms given in Chapter 8 and Chapter 12. As a by-product we also provide good bounds for the problem of estimating vector valued functions.

Applications: Kernel Boosting LP Machines RBF Networks

The generalization performance of learning machines can be bounded in terms of the covering number (see Definition 1.8) of the loss function induced hypothesis class. The necessary tools can be found in Section 1.2.3. In particular Theorem 1.10 states the connection between $\mathcal{N}$ and $R(f)$.

Before going into the actual calculations let us briefly review existing results on this topic. Covering numbers for classes of functions defined via an ℓ_2 constraint were proven for convex combinations of hypotheses by Lee et al. [1996]. Their result, however, was solely based on a theorem of Maurey [1981] ignoring effects of the kernel. Gurvits and Koiran [1997] give a bound for similar settings and Bartlett [1998] proved bounds on the fat shattering dimension based on a weight constraint of Multilayer Perceptrons. See also [Anthony and Bartlett, 1999, Sec. 14.6] for an overview.

In addition to the techniques pointed out in Chapter 1 we will have to introduce entropy numbers (the functional inverse of covering numbers) along with a number of background results in Section 21.2. Next we present a result concerning arbitrary convex combinations of a parametrized family of functions (Section 21.3).

In Section 21.4 we show how one may exploit the geometry of base hypothesis classes H induced by kernels to obtain bounds on $\mathcal{N}(\epsilon, \mathrm{co}\,(H))$ (the ϵ-covering numbers of the convex hull of H) that are better than those obtainable via general results (such as those by Carl et al. [1999]) which are solely in terms of $\mathcal{N}(\epsilon, H)$.

Finally in Section 21.5 we show how to apply the reasoning based on linear operators and their entropy numbers to classes of functions that certainly cannot be expressed in terms of a single linear operator — the most interesting for learning applications being multilayer perceptrons.

All of the proofs are in the appendix. Also in the appendix is an illustration of the difficulty in using p-convex combinations, with $p > 1$, when there are an infinite number of terms (when $p = 2$, this corresponds to traditional weight decay in the limit of an infinite number of nodes).

21.2 Tools from Functional Analysis

As already pointed out in the introduction, our aim is to provide good bounds on $\mathcal{N}$. While direct computation of the latter is often quite difficult, the use of its functional inverse, the so called entropy number ϵ_n is more amenable to practical analysis.

Definition 21.1 Entropy Numbers

Entropy Numbers

Denote by $U_{\mathcal{A}}$ the unit ball in a metric space $\mathcal{A} = (\mathcal{A}, d)$. The n-th entropy number $\epsilon_n(A) = \epsilon_n(A, d)$ of a set $A \subset \mathcal{A}$ with respect to the metric d is defined as the minimum radius ϵ of balls such that there exist $a_1, \ldots, a_n \in \mathcal{A}$ with $A \subset \bigcup_{i=1}^{n} \epsilon U_{\mathcal{A}} + a_i$.

If $\mathcal{A}$ and $\mathcal{B}$ are normed spaces (e.g., Banach spaces), the entropy number $\epsilon_n(T)$ of an operator $T : \mathcal{A} \to \mathcal{B}$ is defined as the entropy number of the image of the unit ball, i.e., $\epsilon_n(T) := \epsilon_n(T(U_{\mathcal{A}}))$ and the ϵ-covering of $T(U_{\mathcal{A}})$ is with respect to the metric of the space $\mathcal{B}$. We sometimes write $\epsilon_n(T, \mathcal{B})$ to make the metric involved explicit.

By construction ϵ_n is the functional inverse of $\mathcal{N}(\epsilon)$; hence if we can view the class of functions used by a learning machine as generated by applying some operator to a unit ball in some space, we will be able to bound the covering numbers of the machine in terms of the entropy numbers of the operator. If $\mathcal{A}$ and $\mathcal{B}$ are Banach spaces, we will denote by $\mathfrak{L}(\mathcal{A}, \mathcal{B})$ the set of all bounded linear operators mapping from $\mathcal{A}$ to $\mathcal{B}$.

For some learning machines one needs bounds on entropy numbers of convex hulls in terms of the entropy numbers of the base model class. In this chapter we

will demonstrate the difference in scaling between general statements on convex combinations and the improvement that can be obtained for the special class of kernel functions by explicitly exploiting the kernel map. We can use a special case of [Carl et al., 1999, Corollary 4.5].

Proposition 21.2 Entropy numbers of Convex Hulls
For all Banach spaces $\mathcal{A}$ and all precompact subsets $A \subset \mathcal{A}$ satisfying the bound

$$\epsilon_n(A) \leq cn^{-\frac{1}{p}} \text{ with } c, p > 0, n \in \mathbb{N} \tag{21.1}$$

there exists a constant $\rho(p)$ such that for all $n \in \mathbb{N}$,

$$\epsilon_{2^n}(\text{co}\,(A)) \leq c\rho(p)n^{-\frac{1}{p}}, \tag{21.2}$$

where $\text{co}\,(A) = \bigcup_{n=1}^{\infty}\{\sum_{i=1}^{n}\alpha_i a_i | a_i \in A,\ \sum_i |\alpha_i| \leq 1\}$ is the (symmetric absolute) convex hull of A.

Convex Hulls

This result will be useful in computing the entropy number of convex combinations, once the entropy number of the base class has been determined. The following proposition which follows directly from volume considerations addresses the latter problem.

Compact and finite dimensional

Proposition 21.3 Compact sets
Given a p-dimensional Banach space $\mathcal{A}$ and a compact set $\Gamma \subset \mathcal{A}$ there exists a constant $c(\Gamma, \mathcal{A}) > 0$ such that the entropy number satisfies

$$\epsilon_n(\Gamma) \leq c(\Gamma, \mathcal{A})\text{vol}\,(\Gamma)^{\frac{1}{p}}n^{-\frac{1}{p}}. \tag{21.3}$$

The constants depend on the geometrical properties of the space, e.g., whether Γ is a box or a ball. Finally we need another bound to take advantage of the fact that we only evaluate a function $f \in F$ on an m-sample. This can be achieved by Maurey's theorem (see [Carl, 1985]). We state a special case applicable to Hilbert spaces, since that is all we need in the present paper.

Hilbert spaces

Proposition 21.4 Maurey, Carl
Let $m \in \mathbb{N}$, H a Hilbert space and let $S \in \mathfrak{L}(H, \ell_\infty^m)$ be a linear operator. Then there exists a constant c such that

$$\epsilon_{2^n}(S) \leq c\|S\|\left(n^{-1}\log\left(1 + \tfrac{m}{n}\right)\right)^{\frac{1}{2}}. \tag{21.4}$$

Vector-valued sequence spaces

We will make use of vector-valued sequence spaces. If $\mathcal{X}$ is a normed space with norm $\|\cdot\|_{\mathcal{X}}$, and $x = (x_1, \ldots, x_m)^T \in \mathcal{X}$, then $\|x\|_{\ell_p^m(\mathcal{X})} := \|(\|x_1\|_{\mathcal{X}}, \ldots, \|x_m\|_{\mathcal{X}}\|_{\ell_p^m}$ where $\|\cdot\|_{\ell_p^m}$ is the traditional ℓ_p^m norm, $\|z\|_{\ell_p^m} = \left(\sum_{i=1}^m |z_i|^p\right)^{1/p}$. In particular denote by $\ell_p^q(\ell_r^s)$ a "mixed" norm acting on $\mathbb{R}^{q\cdot s}$. (See [Diestel et al., 1995, p.32].)

Corollary 21.5 Bounds for $\ell_\infty^m(\ell_1^d)$ spaces
Let $m, d \in \mathbb{N}$, H a Hilbert space, and let $S \in \mathfrak{L}(H, \ell_\infty^m(\ell_1^d))$. Then there exists a constant $c = c(d)$ such that

$$\epsilon_{2^n}(S) \leq c\|S\|\left(n^{-1}\log\left(1 + \tfrac{md}{n}\right)\right)^{\frac{1}{2}}. \tag{21.5}$$

Since $\ell_\infty^m(\ell_1^d)$ is norm equivalent to ℓ_∞^{md} (albeit with a constant depending on d), the corollary can be seen to follow directly from Proposition 21.4.

Finally one needs methods of combining these bounds, e.g., when mapping sets whose entropy numbers are bounded into another space with operators that might restrict the model class even more. The following proposition from [Carl and Stephani, 1990] is very useful.

Product inequality

Proposition 21.6 Products of operators
Suppose $\mathcal{A}, \mathcal{B}, \mathcal{C}$ are Banach spaces and $R \in \mathfrak{L}(\mathcal{A}, \mathcal{B})$ and $S \in \mathfrak{L}(\mathcal{B}, \mathcal{C})$. If $n_1, n_2 \in \mathbb{N}$ and $n \geq n_1 n_2$ then the entropy numbers of $RS : \mathcal{A} \to \mathcal{C}$ satisfy

$$\epsilon_n(RS) \leq \epsilon_{n_1}(R)\epsilon_{n_2}(S). \tag{21.6}$$

A simple variation on this standard result is the following.

Proposition 21.7
Suppose $\mathcal{A}$ and $\mathcal{B}$ are Banach spaces, $V \subseteq \mathcal{A}$ and $S \in \mathfrak{L}(\mathcal{A}, \mathcal{B})$. Then for all $n_1, n_2, n \in \mathbb{N}$ such that $n \geq n_1 n_2$,

$$\epsilon_n(S(V)) = \epsilon_n(S(V), \mathcal{B}) \leq \epsilon_{n_1}(V, \mathcal{A})\epsilon_{n_2}(S(U_\mathcal{A}), \mathcal{B}) = \epsilon_{n_1}(V)\epsilon_{n_2}(S). \tag{21.7}$$

This is needed if V cannot be seen as generated by a linear operator. In some cases, products of operators will not be sufficient, especially if the overall function class cannot be viewed as generated by a single linear operator itself. However, the effect of a nonlinear operator can be seen as being contained in the union of several linear ones, as the following proposition shows.

Concatenation Result

Proposition 21.8 Sets of operators
Denote by $\mathcal{W}, \mathcal{Y}$ Banach spaces, S a linear operator $S : \mathcal{W} \to \mathcal{Y}$, $\mathfrak{L}(\mathcal{W}, \mathcal{Y})$ the space of such operators and $\mathcal{S} \subset \mathfrak{L}(\mathcal{W}, \mathcal{Y})$. Consider the pseudo norm[1] on $\mathfrak{L}(\mathcal{W}, \mathcal{Y})$ induced by a set $W \subset \mathcal{W}$ in a fashion similar to the standard operator norm on $\mathfrak{L}(\mathcal{W}, \mathcal{Y})$:

$$\|S\|_W := \sup_{w \in W} \|Sw\|_\mathcal{Y}. \tag{21.8}$$

Let

$$\mathcal{S}W := \bigcup_{S \in \mathcal{S}} SW. \tag{21.9}$$

---

1. It is easy to check that $\|S\|_W$ is a pseudo norm. In fact we have

$$\|S + S'\|_W = \sup_{w \in W} \|(S + S')w\|_\mathcal{Y} \leq \sup_{w \in W} \|Sw\|_\mathcal{Y} + \sup_{w \in W} \|S'w\|_\mathcal{Y} = \|S\|_W + \|S'\|_W$$
$$\|\lambda S\|_W = \sup_{w \in W} \|\lambda S w\|_\mathcal{Y} = |\lambda| \sup_{w \in W} \|Sw\|_\mathcal{Y} = |\lambda| \|S\|_W$$
$$\|S\|_W = \sup_{w \in W} \|Sw\|_\mathcal{Y} \geq \|Sw^*\|_\mathcal{Y} \geq 0$$

Then for $n, n' \in \mathbb{N}$

$$\begin{aligned}\epsilon_{n\cdot n'}(\mathcal{S}W) &:= \epsilon_{n\cdot n'}(\mathcal{S}W, \|\cdot\|_W) := \epsilon_{n\cdot n'}\left(\cup_{S\in\mathcal{S}} SW, \mathcal{Y}\right) \\ &\leq \epsilon_n(\mathcal{S}, \|\cdot\|_W) + \sup_{S\in\mathcal{S},\mathcal{Y}} \epsilon_{n'}(SW).\end{aligned} \tag{21.10}$$

In particular for W being the unit ball, i.e., $W = U_{\mathcal{W}}$ the metric on $\mathfrak{L}(\mathcal{W}, \mathcal{Y})$ reduces to the standard operator norm and we have

$$\epsilon_{n\cdot n'}(\mathcal{S}U_{\mathcal{W}}) \leq \epsilon_n(\mathcal{S}) + \sup_{S\in\mathcal{S}} \epsilon_{n'}(S) \tag{21.11}$$

This proposition will become very useful in the case of concatenations of nonlinear estimators such as in multilayer perceptrons (see Section 21.8 for a proof). There, each subsequent layer can be represented in terms of set of operators acting on the output of the previous layer (cf. Section 21.5).

Now that all the basic ingredients have been presented we may proceed by proving bounds for the classes of functions used by practical learning algorithms.

21.3 Convex Combinations of Parametric Families

Vector Valued Convex Combination

Consider the class of functions $\mathrm{co}_\Lambda F$ obtained by an absolute convex combination of some parametric family of basis functions $F = F_\Gamma := \{f_\gamma | f_\gamma : \mathcal{X} \to \mathbb{R} \text{ with } \gamma \in \Gamma\}$:

$$\mathrm{co}_\Lambda F := \left\{ f \;\middle|\; f = \sum_i \alpha_i f_{\gamma_i} \text{ with } \alpha_i \in \mathbb{R}^d,\ \sum_i \|\alpha_i\|_{\ell_1^d} \leq \Lambda, \gamma_i \in \Gamma \right\}. \tag{21.12}$$

Observe elements $f \in \mathrm{co}_\Lambda F$ map $f\colon \mathcal{X} \to \mathbb{R}^d$. Recall the L_∞ norm for functions $f\colon \mathcal{X} \to \mathbb{R}$ is $\|f\|_{L_\infty} = \sup_{x\in\mathcal{X}} |f(x)|$. The $L_\infty(\ell_1^d)$ norm for functions $f\colon \mathcal{X} \to \mathbb{R}^d$ is $\|f\|_{L_\infty(\ell_1^d)} = \sup_{x\in\mathcal{X}} \|f(x)\|_{\ell_1^d}$.

For functions f_γ Lipschitz continuous in their parametrization γ with compact finite dimensional index sets Γ one obtains the following statement.

Proposition 21.9 Convex Combinations in $L_\infty(\ell_1^d)$ spaces
Denote by $\Gamma \subset \mathcal{X}$ a compact p-dimensional index set, and F_Γ the corresponding parametric family with $|f_\gamma(\mathbf{x})| \leq 1$ for all $\mathbf{x} \in \mathcal{X}$ and $f_\gamma \in F_\Gamma$. Moreover denote by $c_L(\Gamma, X)$ a Lipschitz constant satisfying

$$\sup_{\gamma,\gamma'\in\Gamma} \sup_{\mathbf{x}\in\mathcal{X}} |f_\gamma(\mathbf{x}) - f_{\gamma'}(\mathbf{x})| \leq c_L(\Gamma, X)\|\gamma - \gamma'\|. \tag{21.13}$$

Then there exists a positive constant $c(\Gamma, p, \mathcal{X}) > 0$ such that

$$\epsilon_{2^n}(\mathrm{co}_\Lambda F, L_\infty(\ell_1^d)) \leq c(\Gamma, p, \mathcal{X}) c_L(\Gamma, \mathcal{X}) \Lambda n^{-\frac{1}{pd}}. \tag{21.14}$$

Evaluation operator

Next we can bound $\epsilon_n(\mathrm{co}_\Lambda F, \ell_\infty^m(\ell_1^d))$ which corresponds to measuring the richness of $\mathrm{co}_\Lambda F$ on an arbitrary m-sample $X = (\mathbf{x}_1, \ldots, \mathbf{x}_m) \subset \mathcal{X}$. For this purpose we introduce the *evaluation operator* S_X as

$$S_X : L_\infty(\ell_1^d) \to \ell_\infty^m(\ell_1^d) \tag{21.15}$$

$$S_X : f \mapsto (f(\mathbf{x}_1), \ldots, f(\mathbf{x}_m)). \tag{21.16}$$

The first thing to note is that S_X is linear and has norm 1 due to the $L_\infty(\ell_1^d)$ norm. Hence we can apply Proposition 21.7 to bound $\epsilon_n(S_X(\mathrm{co}_\Lambda F), \ell_\infty^m(\ell_1^d))$ by $\epsilon_{n_1}(S_X)\epsilon_{n_2}(\mathrm{co}_\Lambda F)$ with $n_1 n_2 \leq n$. Next we use Propositions 21.4 and 21.9 to bound the terms $\epsilon_{n_1}(S_X)$ and $\epsilon_{n_2}(\mathrm{co}_\Lambda F)$ respectively.[2]

Simple Rate Bound

Proposition 21.10 Convex Combinations in $\ell_\infty^m(\ell_1^d)$ spaces
The entropy number (with respect to the $\ell_\infty^m(\ell_1^d)$ metric) of the Λ-convex combination $\mathrm{co}_\Lambda F$ evaluated at m arbitrary points $X := (\mathbf{x}_1, \ldots, \mathbf{x}_m) \subset \mathcal{X}$ satisfies

$$\begin{aligned} e_n &= \epsilon_{2^n}(S_X \mathrm{co}_\Lambda F, \ell_\infty^m(\ell_1^d)) \\ &\leq \Lambda \tilde{c}(\Gamma, p, X) c_L(\Gamma, X) \inf_{n_1, n_2 \in \mathbb{N}, n_1 + n_2 \leq n} \left(n_1^{-1} \log\left(1 + \tfrac{md}{n_1}\right)\right)^{\frac{1}{2}} n_2^{-\frac{1}{pd}} \end{aligned} \tag{21.17}$$

for some constant $\tilde{c}(\Gamma, p, X) > 1$.

By setting $n_1 = n_2 = \lfloor n/2 \rfloor$ one can check that $e_n = O(n^{-\frac{1}{2}-\frac{1}{pd}})$. Since X was arbitrary, we can thus bound $\sup_{X \in \mathcal{X}^m} \mathcal{N}(\epsilon, (\mathrm{co}_\Lambda F)(X), \ell_\infty^m(\ell_1^d))$ by inverting the bound on $\epsilon_n(\mathrm{co}_\Lambda F)$. Ignoring $\log(m)$ terms, one gets

$$\log \sup_{X \in \mathcal{X}^m} \mathcal{N}(\epsilon, (\mathrm{co}_\Lambda F)(X), \ell_\infty^m(\ell_1^d)) = O\left(\epsilon^{-\frac{2pd}{pd+2}}\right) \tag{21.18}$$

For large p or d this is roughly $O(1/\epsilon^2)$ which is similar to the results obtained in [Gurvits, 1997, Bartlett, 1998] derived using the fat-shattering dimension, a version of Maurey's theorem and the generalization of the Sauer-Shelah-Vapnik-Chervonenkis lemma of Alon et al. [1997]. When $\log(m)$ factors are taken into account the above result is slightly better than those previous results.

Better Bounds via Kernels

As we will show subsequently, one can do much better by exploiting properties of kernels in a more explicit way. The reason we can do better is that we take more account of the geometry of F than just its covering numbers. The fact that information about $\mathcal{N}(\epsilon, F)$ alone can not provide tight bounds on $\mathcal{N}(\epsilon, \mathrm{co}_\Lambda F)$ has been observed previously by Talagrand [1993]. The easiest way to see that bounds such as Proposition 21.2 can not be always tight is to observe that $\mathrm{co}\,\mathrm{co}\, F = \mathrm{co}\, F$, but the bound of the proposition would not even apply in that case.

21.4 Convex Combinations of Kernels

Better bounds on ϵ_n can be obtained for convex combinations of kernels. Specifically we are interested in computing $\mathcal{N}(\epsilon, \mathrm{co}\, F)$ when $F = \{\mathbf{x} \mapsto k(\mathbf{x}, \mathbf{x}_1) | \mathbf{x}_1 \in X\}$. In order to do this we take the point of view in [Williamson et al., 1998]: the hypothesis class is considered as the image of a linear operator.

2. Note that $\epsilon_n(S_X(\mathrm{co}_\Lambda F)) = \epsilon_n((\mathrm{co}_\Lambda F)(X))$.

21.4.1 Feature Space

Generalized kernels

We use the definitions of Section 1.3.2, however with a deviation from the conditions on k imposed by Mercer's Theorem (Th. 1.16). Specifically we only require $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ to be a bounded symmetric function in its arguments (no positivity needed). Note that the requirement is similar to the one in Chapter 8.

Moreover we assume that there exists an expansion of k into the eigensystem $(\lambda_i, \psi_i(\mathbf{x}))$ of the corresponding symmetric integral operator (cf. (1.65))

$$Tf(\mathbf{x}) := \int_{\mathcal{X}} k(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{y} \tag{21.19}$$

such that (cf. (1.67))

$$k(\mathbf{x}, \mathbf{y}) = \sum_i \lambda_i \psi_i(\mathbf{x}) \psi_i(\mathbf{y}). \tag{21.20}$$

We will require that k induces a trace-class operator, i.e., that $\sum_i |\lambda_i|$ is finite, and that moreover there exists a constant C_k such that

$$\sup_{i \in \mathbb{N}} \sup_{\mathbf{x} \in \mathcal{X}} |\psi_i(\mathbf{x})| \leq C_k. \tag{21.21}$$

The latter is standard for Mercer kernels [Mercer, 1909], however for general symmetric operators this need not automatically be the case. Consequently the class of admissible functions is significantly larger than the one suitable for SV machines. For instance, B_n spline kernels (i.e., $n+1$ times convolutions of the unit interval) of arbitrary order $n \in \mathbb{N}$ can be used, whereas SV machines would only allow spline kernels of odd order. The crucial point in dealing with convex combinations of kernels is that elements of

Extending the Kernel Trick

$$\mathrm{co}_\Lambda F = \left\{ f : \mathcal{X} \to \mathbb{R}^d \,\middle|\, f(\mathbf{x}) = \sum_i \alpha_i k(\mathbf{x}_i, \mathbf{x}), \alpha_i \in \mathbb{R}^d, \ \sum_i \|\alpha_i\|_{\ell_1^d} \leq \Lambda, \ \mathbf{x}_i \in \mathcal{X} \right\} \tag{21.22}$$

still can be written as a dot product in some feature space. (By definition of $\mathrm{co}_\Lambda F$ we have $\sum_i |\alpha_{ij}| \leq \Lambda_j$ with $\sum_j \Lambda_j \leq \Lambda$.) This is done by setting

$$f(\mathbf{x}) = \sum_{i=1}^m \alpha_i k(\mathbf{x}_i, \mathbf{x}) \tag{21.23}$$

$$= \sum_{i=1}^m \alpha_i \sum_j \lambda_j \psi_j(\mathbf{x}_i) \psi_j(\mathbf{x}) = ((\mathbf{w}_1, \Phi(\mathbf{x})), \ldots, (\mathbf{w}_d, \Phi(\mathbf{x}))). \tag{21.24}$$

Here $\mathbf{w}_j$ and $\Phi(\mathbf{x})$ are defined as follows (for SV kernels this definition coincides with the standard form derived from Mercer's theorem (1.68)):

$$\Phi(\mathbf{x}) := \left(\sqrt{|\lambda_1|} \psi_1(\mathbf{x}), \sqrt{|\lambda_2|} \psi_2(\mathbf{x}), \ldots \right) \tag{21.25}$$

$$\mathbf{w}_j := \left(\sqrt{|\lambda_1|} \operatorname{sgn}(\lambda_1) \sum_{i=1}^m \alpha_{ij} \psi_1(\mathbf{x}_i), \sqrt{|\lambda_2|} \operatorname{sgn}(\lambda_2) \sum_{i=1}^m \alpha_{ij} \psi_2(\mathbf{x}_i), \ldots \right) \tag{21.26}$$

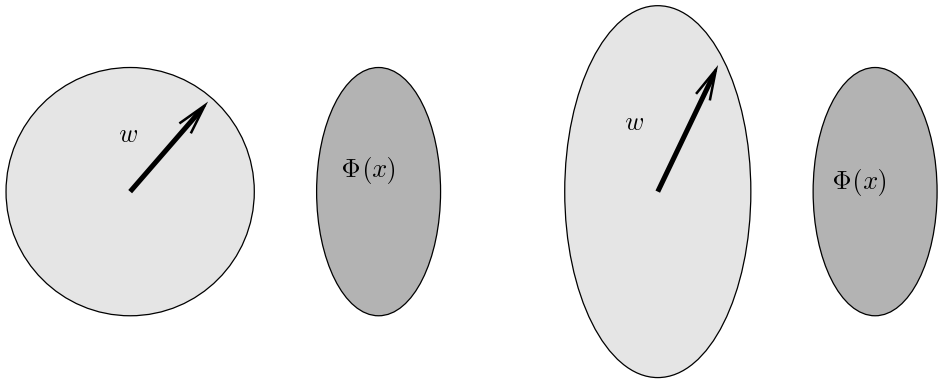


Figure 21.1 Left: In the SV case the weight vector $\mathbf{w}$ is contained in a ball of some (given) radius and the data lies inside some hyperellipsoid. Right: In the convex combination algorithms the weight vector $\mathbf{w}_j$ is contained in a scaled version of the convex hull of the data $\Phi(\mathcal{X})$, e.g., a hyperellipsoid of identical shape but different size.

It is understood that α_{ij} denotes the j-th component of α_i. From the assumptions above one can see that in analogy to [Williamson et al., 1998] again $\bigcup_{\mathbf{x}\in\mathcal{X}} \pm\Phi(\mathbf{x})$ is contained inside a box B with sidelengths $2C_k\sqrt{\lambda_i}$. Hence also $\mathbf{w}_j$ is contained in a scaled version $\Lambda_j B$ since it is a convex combination of elements from B and moreover by construction $\sum_j \Lambda_j \leq \Lambda$. This restriction of $\mathbf{w}_j$ is exactly the property we take advantage of to derive the new bounds.

21.4.2 Scaling and Evaluation Operators

Rather than dealing with parallelepipeds B we will use hyperellipsoids $\mathcal{E}$ for convenience, since the latter can be seen to have been generated by scaling the unit ball in ℓ_2 according to some operator A. With slight abuse of notation, the situation we construct is summarised in the following diagram.

Scaling Operator

$$\begin{array}{ccccc} \mathcal{X} & \xrightarrow{\Phi} & \Phi(\mathcal{X}) & \xrightarrow{A^{-1}} & U_{\ell_2} \\ & & \cap & \swarrow_{A} & \\ & & \mathcal{E} & & \end{array} \tag{21.27}$$

That is, we seek an operator $A : \ell_2 \to \ell_2$ such that $AU_{\ell_2} =: \mathcal{E} \supseteq \Phi(\mathcal{X})$ which implies $A^{-1}\Phi(\mathcal{X}) \subseteq U_{\ell_2}$. This can be ensured by constructing A such that

$$A\colon (\mathbf{x}_j)_j \mapsto (R_A \cdot a_j \cdot \mathbf{x}_j)_j \text{ where } a_j \in \mathbb{R}^+ \tag{21.28}$$

with $R_A := C_k \|(\sqrt{|\lambda_j|}/a_j)_j\|_{\ell_2}$ where C_k is the constant from (21.21). Hence the situation (see Figure 21.1) is quite similar to the SV case [Williamson et al., 1998]. The mapped data is contained inside some hyperellipsoid. The weight vectors $\mathbf{w}_j$, however, are constrained to a ball in the SV case and to a hyperellipsoid $\mathrm{diag}(\Lambda_1, \Lambda_2, \ldots)\mathcal{E}$ of the same shape as the original data in the case of convex combinations. This means that while in SV machines capacity is allocated equally

Two Ellipsoids

along all directions, in the present case much capacity is allocated in those directions where the data is spread out a lot and little capacity where there is little spread. Since $f(\mathbf{x}) \in \mathbb{R}^d$ one has to apply the scaling operator A for each output dimension separately, i.e., one effectively has to apply the operator A_d (in a similar fashion to [Smola et al., 1999]) with

Multiple Scaling

$$A_d : \ell_2(\ell_2^d) \to \ell_2(\ell_2^d) \text{ with } A_d := \underbrace{A \times A \times \ldots \times A}_{d-\text{times}}. \tag{21.29}$$

Before carrying out the exact calculations we define an appropriate evaluation operator $S_{\Phi(X)}$. We set

Multiple Evaluation

$$S_{\Phi(X)} : \ell_2(\ell_2^d) \to \ell_\infty^m(\ell_1^d) \tag{21.30}$$

$$S_{\Phi(X)} : (\mathbf{w}_1, \ldots \mathbf{w}_d) \mapsto \begin{pmatrix} ((\Phi(\mathbf{x}_1), \mathbf{w}_1), \ldots, (\Phi(\mathbf{x}_1), \mathbf{w}_d)) \\ \vdots \\ ((\Phi(\mathbf{x}_m), \mathbf{w}_1), \ldots, (\Phi(\mathbf{x}_m), \mathbf{w}_d)) \end{pmatrix}. \tag{21.31}$$

This operator evaluates the estimate f on the dataset X, and it is precisely the entropy number of the image of $S_{\Phi(X)}$ we are seeking. The present considerations lead to the following theorem for convex combinations of kernels in analogy to the results in [Williamson et al., 1998].

21.4.3 Bounds on Entropy Numbers

Theorem 21.11 Bounds for Linear Programming Machines
Denote by k a symmetric bounded kernel, let Φ be induced via (21.25) and let $S_{\Phi(X)}$ be given by (21.30). Moreover let A be defined by (21.28) and A_d by (21.29). Then the entropy numbers of $\mathrm{co}_\Lambda F$ satisfy the following inequalities: For $n, t \in \mathbb{N}$,

$$\epsilon_n(\mathrm{co}_\Lambda F) \leq c\Lambda \|A_d\|^2 \log^{-1/2}(n) \log^{1/2}\left(1 + \tfrac{dm}{\log n}\right) \tag{21.32}$$

$$\epsilon_n(\mathrm{co}_\Lambda F) \leq \Lambda \epsilon_n(A_d^2) \tag{21.33}$$

$$\epsilon_{nt}(\mathrm{co}_\Lambda F) \leq c\Lambda \log^{-1/2}(n) \log^{1/2}\left(1 + \tfrac{dm}{\log n}\right) \epsilon_t(A_d^2) \tag{21.34}$$

where c is a constant as defined in Corollary 21.5.

This result (and also its proof) is a modified combination of the results in [Williamson et al., 1998, Smola et al., 1999]; the key difference is that the weight vector is constrained to a different set and that is why the operator A_d appears twice.

It remains to bound the entropy number of A_d. We use a slight variation on a result from [Smola et al., 1999].

Corollary 21.12 Entropy numbers for the vector valued case
Let k be a kernel which induces a trace-class integral operator and satisfies (21.21). Let A be defined by (21.28) and A_d by (21.29). Then

$$\begin{aligned}&\epsilon_n(A_d\colon \ell_2(\ell_2^d) \to \ell_2(\ell_2^d))\\ \leq &\inf_{(a_s)_s:\left(\frac{\sqrt{\lambda_s}}{a_s}\right)_s \in \ell_2} \sup_{j\in\mathbb{N}} 6C_k\sqrt{d}\left\|\left(\frac{\sqrt{\lambda_s}}{a_s}\right)_s\right\|_{\ell_2} n^{-\frac{1}{j\cdot d}}(a_1 a_2 \cdots a_j)^{\frac{1}{j}}.\end{aligned} \tag{21.35}$$

21.4.4 Applications to Kernel Functions

Although the above results seem rather abstruse and complex, it turns out they can be applied without too much pain. By using arguments as in [Williamson et al., 1998] (subsequently further simplified by Guo et al. [1999]) one can explicitly compute the entropy numbers of the A_d operator. The following two propositions follow immediately from their counterparts for the case of SV regularization.

Proposition 21.13 Polynomial Decay
Let k be a symmetric kernel with eigenvalues $|\lambda_j| = O(j^{-(\alpha+1/2)})$ and $\alpha > 0$. Then

$$\epsilon_n(A_d^2\colon \ell_2(\ell_2^d) \to \ell_2(\ell_2^d)) = O(\ln^{-\alpha} n). \tag{21.36}$$

This result can be seen as follows. As A is a diagonal scaling operator, the scaling factors of A^2 are simply those of A squared, i.e., decaying twice as fast. Moreover, the dimensionality of the output does not change the rate of decay in terms of the eigenvalues λ_i except for a constant factor. Comparing the result with its SV counterpart in [Williamson et al., 1998] shows that the condition on the eigenvalues was changed from $i^{-(\alpha/2+1)}$ into $i^{-(\alpha+1/2)}$. The conclusions and the method of proving this, however, remain unchanged. A similar result can be stated for exponentially decaying eigenvalues of k.

Since the eigenfunctions of a translation invariant kernel are the traditional Fourier bases, the eigenvalues can be determined in terms of Fourier transform coefficients. We then have:

Proposition 21.14 Polynomial Exponential Decay in $\mathbb{R}^d$
For translation invariant kernels, $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}')$ in $\mathbb{R}^d \times \mathbb{R}^d$ with Fourier transform satisfying $\ln|F[k](\omega)| \leq O(\|\omega\|^p)$ with $p > 0$ and corresponding operator A_d one has

$$\ln \epsilon_n^{-1}(A_d^2\colon \ell_2(\ell_2^d) \to \ell_2(\ell_2^d)) = O(\ln^{\frac{p}{p+d}} n). \tag{21.37}$$

Analogous results hold for the other propositions obtained in [Williamson et al., 1998]. Note that whereas in Proposition 21.13 an improvement of the rates in n was achievable (over those in [Williamson et al., 1998]), in Proposition 21.14 no such thing happened since the bound is in terms of $\ln \epsilon_n$ instead of ϵ_n. The constants would be quite different though.

Of course the above considerations only indicate that the class of functions implemented by linear programming machines is smaller (in the sense of smaller covering numbers) than that implemented by traditional support vector machines. This affects the bound on generalization error, but does not imply that error will be smaller: for some problems traditional SV machines may achieve smaller error.

The point is that the capacity is distributed *differently* among the class of kernel expandable functions, i.e., a different structure (in the sense of structural risk minimization) is chosen. More emphasis is put on the first eigenfunctions of the kernel. If one has experimental evidence that this might be useful (say, e.g., from compression experiments [Schölkopf et al., 1999b]), one should consider using such a regularizer.

Examples of kernels with rapid decay of the first eigenvalues are Gaussian RBF-kernels $k(\mathbf{x}, \mathbf{x}') = \exp(-\|\mathbf{x} - \mathbf{x}'\|^2)$ $(p = 2)$, or the "damped harmonic oscillator" kernel $k(\mathbf{x}, \mathbf{x}') = 1/(1 + \|\mathbf{x} - \mathbf{x}'\|)$ $(p \geq 1)$. Since $\epsilon_n(A_d^2)$ enters into the overall bound the overall covering numbers can be smaller than in the SV case where we have to bound $\epsilon_n(A_d)$.

21.5 Multilayer Networks

Whilst the techniques presented so far provide efficient tools for dealing with the capacity of sets generated by linear operators in some Hilbert spaces, many practical cases fail to satisfy these assumptions (e.g., multilayer perceptrons, rbf-networks, or combinations thereof). However, many of the latter ways of representing functions can be seen as generated by (nonlinear) concatenations of linear operators.

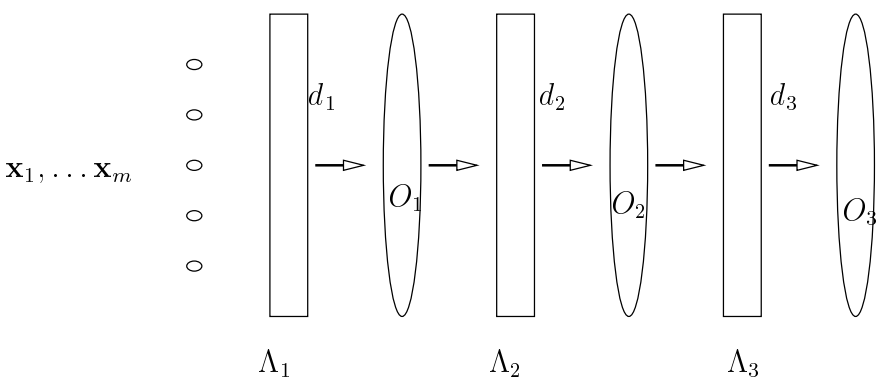


Figure 21.2 Structure of a Multilayer Network. Data is fed in on the left hand side. Each processing layer maps a (with respect to this layer) fixed input into a set of outputs via the evaluation operator S_X. Thus the possible outputs of a layer consist of the union of outputs for all different evaluation operators S_X. The output dimensionality is denoted by d_i, the size of the model class per layer by Λ_i.

Figure 21.2 depicts a multilayer network. The i first layers create, for fixed input, a *set* of outputs $O_i \subset \ell_\infty^m(\ell_1^{d_i})$ when the constraint Λ_i is imposed on the model class of the corresponding layer by requiring that the sum of the absolute values of the weights in the ith layer is Λ_i. The outputs O_i can be seen as generated by a class of operators $\mathcal{S}_i$ defined as the set of all possible evaluation operators

$$\mathcal{S}_i := \bigcup_{X \in O_{i-1}} S_{\Phi(X)} \text{ with } i > 1 \tag{21.38}$$

For $i = 1$ we set $\mathcal{S}_1 := \{S_{\Phi(X)}\}$ where X is the actual training data. Thus at each layer we have a situation as in Proposition 21.8.

We need to compute the entropy number of $\mathcal{S}_i$ in order to compute the entropy number of O_i. The following proposition uses the connection between $\epsilon_n(O_{i-1})$ and $\epsilon_n(\mathcal{S}_i)$. Moreover, in order to apply our result to Regularization Network type Multilayer Perceptrons it pays off to have a specific connection between $\mathcal{S}_i$ and O_i for Tikhonov regularizers as well (see (21.42) and (21.43)).

Proposition 21.15 Entropy numbers for classes of operators
Let $k(\mathbf{x}, \mathbf{x}')$ be a kernel with Lipschitz constant l_k, i.e.,

$$|k(\mathbf{x}, \mathbf{x}') - k(\mathbf{x}, \mathbf{x}'')| \leq l_k \|\mathbf{x}' - \mathbf{x}''\| \text{ for all } \mathbf{x}, \mathbf{x}', \mathbf{x}'' \in \mathcal{X}, \tag{21.39}$$

where $\mathcal{X}$ is an index set with entropy number $\epsilon_n(\mathcal{X})$, and $\mathcal{S}_i$ the set of operators as defined in (21.38). Then the following bound holds: For W defined according to (21.12), i.e.,

$$W_\Lambda := \left\{ (\mathbf{w}_1, \ldots, \mathbf{w}_{d_o}) \,\middle|\, \mathbf{w}_j = \textstyle\sum_j \alpha_{ij} \Phi(\mathbf{x}_i) \text{ where } \alpha_{ij} \in \mathbb{R}, \sum_{i,j} |\alpha_{ij}| \leq \Lambda \right\} \tag{21.40}$$

and the $\ell_\infty^m(\ell_1^{d_{i-1}})$ on $\mathcal{X}$ and the $\ell_1^{d_i}$ metric on $\mathcal{Y}$ we obtain (recall the definition (21.9))

$$\epsilon_n(\mathcal{S} W_\Lambda) \leq \Lambda l_k \epsilon_n(\mathcal{X}). \tag{21.41}$$

Moreover for W defined as

$$W_\Lambda := \left\{ (\mathbf{w}_1, \ldots, \mathbf{w}_{d_o}) \,\middle|\, \sum_{i=1}^{d_o} \|\mathbf{w}_i\|^2 \leq \Lambda \right\} \tag{21.42}$$

and the mixed Euclidean metric on both $\mathcal{X}$ and $\mathcal{Y}$ respectively, i.e., $\ell_\infty^m(\ell_2^d)$, we have

$$\epsilon_n(\mathcal{S}) \leq \sqrt{2\Lambda l_k \epsilon_n(\mathcal{X})}. \tag{21.43}$$

Now we can just go and daisy-chain the separate layers and repeatedly apply Proposition 21.8. For simplicity we will only carry out this calculation for MLPs with a convexity constraint. The second case (via (21.43)) is straightforward. We obtain the following corollary.

Corollary 21.16 Entropy Numbers for Multilayer Perceptrons
For an l layer network MLP as in Figure 21.2 we obtain (set $F := F_1$)

$$\epsilon_n(\text{MLP}) \leq \sum_{i=1}^{l} \epsilon_{n_i}(F)\, l_k^{l-i} \prod_{j=i}^{l} \Lambda_j \tag{21.44}$$

where $n \geq \prod_{i=1}^{l} n_i$.

For the sake of simplicity we assumed that all layers are built in the same way. That is why only $\epsilon_n(F)$ appears in the inequality instead of one different F_i for each layer. In order to see the implications of this result we apply it to kernels satisfying the conditions of Proposition 21.14.

Corollary 21.17 MLPs from kernels with rapidly decaying spectra
For Multilayer Perceptrons built from translation invariant kernels, $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}')$ in $\mathbb{R}^d \times \mathbb{R}^d$ with Fourier transform satisfying $\ln |F[k](\omega)| = O(\|\omega\|^p)$ with $p > 0$, corresponding operator A_d, and ℓ_1 type convexity constraint on the weights one has

$$\ln \epsilon_n^{-1}(MLP\colon \ell_\infty^m(\ell_2^{d'}) \to \ell_\infty^m(\ell_2^d)) = O(\ln^{\frac{p}{p+d}} n) \tag{21.45}$$

This can be seen by applying Proposition 21.14 to each single layer, noticing that the finite sample size part completely dominated by the behaviour of the eigenvalues of the kernel and finally applying Corollary 21.16. There, in particular, note that since in (21.44) only the products of the entropy numbers of the individual layers appear, their effect is equal when taking the logarithm of the overall term.

Hence the asymptotic rate of growth of the covering numbers of an MLP built with such smooth kernels is the same as that for a network with a single hidden layer. Consequently from the point of the asymptotic speed of statistical convergence the class of functions of an MLP cannot be effectively more complicated than that of a single hidden layer network.

21.6 Discussion

We showed that linear programming machines do carry out a form of regularization, which is quite different from the regularization of SV machines. Furthermore, by taking advantage of the specific properties of kernels bounds on the covering numbers of the class of functions computed by such machines can be obtained which are better than those which ignore the effect of the kernel. Specifically, for some kernels (e.g., Gaussian RBF) exponentially better rates (Proposition 21.14) than those for arbitrary kernels (Prop 21.10) can be obtained — observe the ln in (21.37). In addition, we showed that one can extend the techniques to classes of functions such as those generated by multilayer RBF networks. The proofs relied on an operator theoretic viewpoint. The slower rates of growth of covering numbers obtained from the LP regularizer of course do not imply that LP machines

perform better (or worse) than SV machines; just that the "size" of the two effective classes differs and thus so do the generalization error bounds obtained via uniform convergence theorems.

In this extended summary we have limited ourselves to outlining how the rate of growth of covering numbers can be determined. For a successful learning algorithm, however, good estimates of the constants (and not only the rates) are crucial. We refer the reader to [Williamson et al., 1998, Guo et al., 1999] for the calculation of tighter bounds, by more carefully evaluating the inf and sup in the bounds on $\epsilon_n(A)$. It would be of some interest to see if such a more refined calculation could be experimentally corroborated.

21.7 Appendix: A Remark on Traditional Weight Decay

One might conjecture that a result similar to that for ordinary convex hulls could be established for q-convex combinations with $q > 1$, i.e.,

$$F_q := \left\{ f \,\middle|\, f = \textstyle\sum_j \alpha_j f_{\gamma_j} \text{ with } \sum_j |\alpha_j|^q \leq 1 \right\} \tag{21.46}$$

(For the sake of simplicity we only consider $\mathcal{Y} = \mathbb{R}$ in this section.) Training large neural networks with weight decay ($q = 2$) is such a case. However, under the assumption of an infinite number of basis functions the conjecture is false. It is sufficient to show that for $q > 1$, F_q is unbounded in L_∞. Consider an infinite index set $I \subset \Gamma$ for which, for some other set M of nonzero measure and some constant $\kappa > 0$

$$f_\gamma(\mathbf{x}) \geq \kappa \quad \text{for all} \quad f_\gamma \in I, \mathbf{x} \in M. \tag{21.47}$$

An example is $f_\gamma(\mathbf{x}) = e^{-(\mathbf{x}-\gamma)^2}$ for which any compact sets I, M satisfy (21.47). Obviously

$$f(\mathbf{x}) := \sum_j \alpha_j f_{\gamma_j}(x) \geq \kappa \sum_j \alpha_j \text{ for } \alpha_j \geq 0, \gamma_j \in I, \mathbf{x} \in M. \tag{21.48}$$

For $n \in \mathbb{N}$, let $\hat{f}_n := \sum_{j=1}^n n^{-1/q} f_{\gamma_i}$. By construction, the ℓ_q^n norm of the coefficients equals 1, however $\hat{f}_n(x) \geq \kappa n^{1-1/q}$ for all $x \in M$. Thus $\lim_{n\to\infty} \|\hat{f}_n\|_{L_\infty} = \infty$ and therefore F_q contains unbounded elements for $q > 1$, which leads to infinitely large covering numbers for F_q. Thus F_q with $q > 1$ is not a suitable choice as a hypothesis class (in the absence of further regularization).

This leads to the question why, despite the previous reasoning, weight decay has been found to work in practice. One reason is that in standard neural networks settings the number of basis functions is limited (either by construction, via some penalty term, etc.), thus the above described situation might not occur. Secondly, e.g., in rbf-networks, a clustering step for finding the centers is inserted before training the final weights. This means that the basis functions are sufficiently different from each other — observe that the similarity of some basis functions was explicitly exploited in the counterexample above.

Finally, also by the distance of the centers of the basis functions (thus of their peaks), penalization with a diagonal matrix is not too different from penalization via a kernel matrix (provided the widths of the basis functions is equal, and not significantly larger than the distance between the centers) — the main diagonal elements will be 1 and the off diagonal elements rather small, thus an approximation by the unit matrix is not too unrealistic. There exists, however, a case where this reasoning might go wrong in practice. Assume one wants to modify a boosting algorithm in such a way that instead of convex combinations one would like to have p-convex combinations with $p > 1$. After iterating a sufficiently long time the situation described above might occur as the number of basis functions (i.e., weak learners) keeps on increasing with the iterations.

21.8 Appendix: Proofs

Proof **(Proposition 21.7)** Suppose we have an ϵ_{n_1}, n_1 cover of V. Hence we can find $a_1, \ldots, a_{n_1} \in \mathcal{A}$ such that

$$V \subseteq \bigcup_{i=1}^{n_1} \{\epsilon_{n_1} U_{\mathcal{A}} + a_i\}. \tag{21.49}$$

Exploiting the linearity of S yields that

$$S(V) \subseteq \bigcup_{i=1}^{n_1} \{\epsilon_{n_1} S(U_{\mathcal{A}}) + S(a_i)\}. \tag{21.50}$$

Hence, constructing an ϵ_{n_2}, n_2 cover of $S(U_{\mathcal{A}})$ by $b_1, \ldots, b_{n_2} \in \mathcal{B}$ leads to an $\epsilon_{n_1}\epsilon_{n_2}, n_1 n_2$ cover of $S(V)$. Thus we get

$$S(V) \subseteq \bigcup_{i=1}^{n_1} \bigcup_{j=1}^{n_2} \{\epsilon_{n_1}\epsilon_{n_2} b_i + S a_i\} \tag{21.51}$$

which completes the proof. ■

Proof **(Proposition 21.8)** The proof works by constructing the $\epsilon_{n \cdot n'}(\mathcal{S}W)$ cover explicitly.[3] Denote by $\mathcal{S}_\epsilon = \{S_1, \ldots, S_n\} \subset \mathcal{S}$ a set achieving an ϵ cover of $\mathcal{S}$ wrt. the norm induced by W. Moreover denote by $Y_{\epsilon'}(S_i) := \{y_{i1}, \ldots, y_{in'}\}$ an ϵ' cover of $S_i W$. What we have to show is that $\bigcup_{1 \le i \le n} Y_{\epsilon'}(S_i)$, which has cardinality at most $n \cdot n'$, is an $\epsilon + \epsilon'$ cover of $\mathcal{S}W$.

For any $y = S\mathbf{w} \in \mathcal{S}W$ there exists an operator S_i with $\|S\mathbf{w}' - S_i\mathbf{w}'\| \le \epsilon$ for all $\mathbf{w}' \in W$, hence in particular $\|y - S_i\mathbf{w}\| \le \epsilon$ as $\mathbf{w} \in W$. Furthermore there exists a $y_{ij} \in Y_{\epsilon'}(S_i)$ with $\|y_{ij} - S_i\mathbf{w}\| \le \epsilon'$ which leads to $\|y - y_{ij}\| \le \epsilon + \epsilon'$. Finally, such an n cover with ϵ' is always possible for all $S_i W$ since by construction $\epsilon' = \sup_{S \in \mathcal{S}} \epsilon_{n'}(S)$. ■

Proof **(Proposition 21.9)** The first step is to compute an upper bound on $\epsilon_n(F_\Gamma) = \epsilon_n(F_\Gamma, L_\infty(\ell_1^p))$ in terms of the entropy numbers of Γ. By definition we have

$$\|f_\gamma - f_{\gamma'}\| \le c_L(\Gamma, \mathcal{X}) d(\gamma, \gamma') \tag{21.52}$$

and therefore

$$\epsilon_n(F_\Gamma) \le c_L(\Gamma, \mathcal{X}) \epsilon_n(\Gamma). \tag{21.53}$$

---

3. A related approach was taken by Bartlett [1998] to compute the fat shattering dimension of multilayer perceptrons by exploiting a Lipschitz condition. Moreover a similar result was stated in [Haussler, 1992, Lemma 8, pg. 123].

As we are interested in the absolute convex combination in d dimensions, we need to take into account that we have to add in F_Γ for each dimension separately. Let

$$B = \left(\overbrace{(F_\Gamma, \underbrace{0, \ldots, 0}_{d-1 \text{ times}}) \cup \ldots \cup (0, \ldots, 0, F_\Gamma)}^{d \text{ times}} \right) \tag{21.54}$$

denote the base hypothesis class. Clearly if F_Γ is indexed by p-dimensions, B is indexed by pd-dimensions. From Proposition 21.3 we can obtain

$$\epsilon_n(B) \leq c(\Gamma, \mathcal{X}) \mathrm{vol}\,(\Gamma)^{\frac{1}{p}} c_L(\Gamma, \mathcal{X}) n^{-\frac{1}{pd}} \tag{21.55}$$

Now apply Proposition 21.2 to obtain

$$\epsilon_{2^n}(\mathrm{co}_\Lambda\, F) \leq \Lambda \rho(p) c(\Gamma, \mathcal{X}) \mathrm{vol}\,(\Gamma)^{\frac{1}{p}} c_L(\Gamma, \mathcal{X}) \left(\tfrac{1}{n}\right)^{\frac{1}{pd}}. \tag{21.56}$$

Collecting the constants into $c(\Gamma, p, \mathcal{X})$ gives the desired result. ■

Proof **(Theorem 21.11)** The diagram in Equation (21.57) indicates the line of reasoning we use for bounding $\epsilon_n(F_\Lambda)$.

$$\begin{array}{ccccccc} U_{\ell_2(\ell_2^d)} \subset \ell_2(\ell_2^d) & & \xrightarrow{\qquad\qquad T \qquad\qquad} & & & & \ell_\infty^m(\ell_1^d) \\ \Big\downarrow \Lambda & & & & & \nearrow S_{\Phi(X)} & \Big\uparrow S_{A^{-1}\Phi(X)} \\ \Lambda U_{\ell_2} \subset \ell_2(\ell_2^d) & \xrightarrow{A_d} & \Lambda A_d U_{\ell_2(\ell_2^d)} \subset \ell_2(\ell_2^d) & \xrightarrow{A_d} & \Lambda A_d^2 U_{\ell_2} \subset \ell_2(\ell_2^d) & & \end{array} \tag{21.57}$$

Here $T : \ell_2(\ell_2^d) \to \ell_\infty^m(\ell_1^d)$ depicts the linear operator corresponding to F_Λ. In order to bound $\ell_\infty^m(\ell_1^d)$ entropy numbers of the hypothesis class evaluated on an m-sample test set X, one has to bound $\epsilon_n(S_{\Phi(X)}(\Lambda A_d U_{\ell_2(\ell_2^d)}))$, since the weight vectors $(\mathbf{w}_1, \ldots, \mathbf{w}_d)$ will be contained in $\Lambda A_d U_{\ell_2(\ell_2^d)}$. Moreover we have by construction

$$S_{\Phi(X)}(\Lambda A_d U_{\ell_2(\ell_2^d)}) = S_{A^{-1}\Phi(X)}(\Lambda A_d A_d U_{\ell_2}). \tag{21.58}$$

where we used $(\Phi(x), \mathbf{w}_i) = (A^{-1}\Phi(\mathbf{x}), A\mathbf{w}_i)$ which is applicable since f can be represented as a linear functional in some feature space. Using (21.58) and Proposition 21.6 one obtains

$$\begin{array}{rcl} \epsilon_n(S_{\Phi(X)}(\Lambda A_d U_{\ell_2(\ell_2^d)})) & \leq & \Lambda \epsilon_n(S_{A^{-1}\Phi(X)} A_d^2) \\ & \leq & \inf\limits_{n_1, n_2 \in \mathbb{N}, n_1 n_2 \leq n} \epsilon_{n_1}(S_{A^{-1}\Phi(X)}) \epsilon_{n_2}(A_d^2). \end{array} \tag{21.59}$$

Combining the factorization properties obtained above with Proposition 21.6 yields the desired results: by construction, due to the Cauchy-Schwartz inequality $\|S_{A^{-1}\Phi(X)}\| = 1$. Since $S_{A^{-1}\Phi(X)}$ is an operator mapping from a Hilbert space ℓ_2 into an ℓ_∞^m one can use Maurey's theorem (see Proposition 21.4). ■

Proof **(Proposition 21.15)** We have to show that $\sup_{\mathbf{w}\in W_\Lambda} \|S_{\mathbf{x}'}\mathbf{w} - S_{\mathbf{x}''}\mathbf{w}\| \leq \epsilon$ if $\|\mathbf{x}' - \mathbf{x}''\| \leq \epsilon'$. For the case of a linear programming regularizer one has

$$\|S_{\mathbf{x}'}\mathbf{w} - S_{\mathbf{x}''}\mathbf{w}\|_{\ell_\infty^m\left(\ell_1^{d_o}\right)} = \max_{1\leq n\leq m} \sum_{i=1}^{d_o} \left| \left(\sum_j \alpha_{ij} \Phi(\mathbf{x}_j),\, \Phi(\mathbf{x}'_n) - \Phi(\mathbf{x}''_n) \right) \right| \tag{21.60}$$

$$\leq \max_{1\leq n\leq m} \sum_{i=1}^{d_o} \sum_j l_k |\alpha_{ij}|\, \|\mathbf{x}'_n - \mathbf{x}''_n\|_{\ell_1^{d_i}} \tag{21.61}$$

$$\leq l_k \Lambda \|\mathbf{x}' - \mathbf{x}''\|_{\ell_\infty^m\left(\ell_1^{d_i}\right)} \tag{21.62}$$

Assuming that there exists an ϵ cover of X with n points, this automatically generates an $l_k\Lambda\epsilon$ cover of $\mathcal{S}$ with the same number of points, which proves the theorem.

The second part can be shown in a similar manner by exploiting that

$$|(\Phi(\mathbf{x}) - \Phi(\mathbf{x}'), w)|^2 \leq \|\Phi(\mathbf{x}) - \Phi(\mathbf{x}')\|^2 \|w\|^2 \tag{21.63}$$

$$= (k(\mathbf{x},\mathbf{x}) - k(\mathbf{x},\mathbf{x}') + k(\mathbf{x}',\mathbf{x}') - k(\mathbf{x},\mathbf{x}'))\|w\|^2 \tag{21.64}$$

$$\leq 2l_k \|\mathbf{x} - \mathbf{x}'\|_{\ell_2^{d_i}} \|w\|^2 \tag{21.65}$$

Hence we have

$$\|S_{\mathbf{x}}\mathbf{w} - S_{\mathbf{x}'}\mathbf{w}\|^2_{\ell_\infty^m\left(\ell_2^{d_o}\right)} = \max_{1\leq n\leq m} \sum_{i=1}^{d_o} \left|(\mathbf{w}_i, \Phi(\mathbf{x}'_n) - \Phi(\mathbf{x}''_n))\right|^2 \tag{21.66}$$

$$\leq \max_{1\leq n\leq m} \sum_{i=1}^{d_o} 2l_k \|\mathbf{x}'_n - \mathbf{x}''_n\|_{\ell_2^{d_i}} \|\mathbf{w}_i\|^2 \tag{21.67}$$

$$\leq 2l_k\Lambda \|\mathbf{x}' - \mathbf{x}''\|_{\ell_\infty^m\left(\ell_2^{d_i}\right)} \tag{21.68}$$

Again, assuming that there exists an ϵ cover of X with n points, this automatically generates a $\sqrt{2l_k\Lambda\epsilon}$ cover of $\mathcal{S}$ with the same number of points. ■

Acknowledgements

This work was supported in part by grants of the Australian Research Council, the DFG (# Ja 379/51,71,91), and the European Commission under the Working Group Nr. 27150 (NeuroCOLT2). Parts of this work were done while AS and BS were at GMD FIRST.

References

M. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821 – 837, 1964.

A. Albert. *Regression and the Moore-Penrose pseudoinverse*. Academic Press, New York, NY, 1972.

K. S. Alexander. Probability inequalities for empirical processes and a law of the iterated logarithm. *Annals of Probability*, 12:1041–1067, 1984.

D. Allen. The relationship between variable selection and data augmentation and a method for prediction. *Technometrics*, 16:125–127, 1974.

N. Alon, S. Ben-David, N. Cesa-Bianchi, and D. Haussler. Scale–sensitive dimensions, uniform convergence, and learnability. *Journal of the ACM*, 44(4):615–631, 1997.

S. Amari and S. Wu. An information-geometrical method for improving performance of support vector machine classifiers. In D. Willshaw and A. Murray, editors, *Proceedings of ICANN'99*, volume 1, pages 85–90. IEE Press, 1999.

J. A. Anderson. Regression and ordered categorical variables (with discussion). *Journal of the Royal Statistical Society – Series B*, 46:1–30, 1984.

J. A. Anderson and P. R. Philips. Regression, discrimination and measurement models for ordered categorical variables. *Applied Statistics*, 30:22–31, 1981.

J. K. Anlauf and M. Biehl. The adatron: an adaptive perceptron algorithm. *Europhysics Letters*, 10:687 – 692, 1989.

M. Anthony and P. L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337 – 404, 1950.

S. Arora, L. Babai, J. Stern, and Z. Sweedyk. Hardness of approximate optima in lattices, codes, and linear systems. *Journal of Computer and System Sciences*, 54(2):317–331, 1997.

P. L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536, 1998.

P. L. Bartlett, S. R. Kulkarni, and S. E. Posner. Covering numbers for real-valued

function classes. *IEEE Transactions on Information Theory*, 43(5):1721–1724, 1997.

P. L. Bartlett, P. Long, and R. C. Williamson. Fat–Shattering and the Learnability of Real–Valued Functions. *Journal of Computer and System Sciences*, 52(3):434–452, 1996.

P. L. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 43–54, Cambridge, MA, 1999. MIT Press.

E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 1997. (to appear).

K. P. Bennett. Combining support vector and mathematical programming methods for induction. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - SV Learning*, pages 307–326, Cambridge, MA, 1999. MIT Press.

K. P. Bennett and J. A. Blue. A support vector machine approach to decision trees. In *Proceedings of IJCNN'98*, pages 2396–2401, Anchorage, Alaska, 1997.

K. P. Bennett, D. Hui, and L. Auslender. On support vector decision trees for database marketing. Department of Mathematical Sciences Math Report No. 98-100, Rensselaer Polytechnic Institute, Troy, NY 12180, March 1998. http://www.math.rpi.edu/~bennek/.

K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

K. P. Bennett and O. L. Mangasarian. Serial and parallel multicategory discrimination. *SIAM Journal on Optimization*, 4(4):722–734, 1994.

J. O. Berger. *Statistical Decision theory and Bayesian Analysis*. Springer Verlag, New York, 1985.

M. V. Berry. Quantizing a classically ergodic system: Sinai's billiard and the KKR method. *Annals of Physics*, 131:163–216, 1981.

M. Bertero. Regularization methods for linear inverse problems. In C. G. Talenti, editor, *Inverse Problems*. Springer-Verlag, Berlin, 1986.

M. Bertero, T. Poggio, and V. Torre. Ill-posed problems in early vision. *Proceedings of the IEEE*, 76:869–889, 1988.

A. Bertoni, P. Campadelli, and M. Parodi. A boosting algorithm for regression. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Proceedings ICANN'97, Int. Conf. on Artificial Neural Networks*, volume V of *LNCS*, pages 343–348, Berlin, 1997. Springer.

D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, MA, 1995.

C. Blake, E. Keogh, and C. J. Merz. UCI repository of machine learning databases,

1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

V. Blanz, B. Schölkopf, H. Bülthoff, C. Burges, V. Vapnik, and T. Vetter. Comparison of view-based object recognition algorithms using realistic 3D models. In C. von der Malsburg, W. von Seelen, J. C. Vorbrüggen, and B. Sendhoff, editors, *Artificial Neural Networks — ICANN'96*, pages 251 – 256, Berlin, 1996. Springer Lecture Notes in Computer Science, Vol. 1112.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, July 1992. ACM Press.

H. Bourlard and N. Morgan. A continuous speech recognition system embedding MLP into HMM. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 186–193. Morgan Kaufmann, San Mateo, CA, 1990.

P. S. Bradley and O. L. Mangasarian. Feature selection via concave minimization and support vector machines. In J. Shavlik, editor, *Machine Learning Proceedings of the Fifteenth International Conference(ICML '98)*, pages 82–90, San Francisco, CA, 1998. Morgan Kaufmann.

E. J. Bredensteiner. *Optimization Methods in Data Mining and Machine Learning*. PhD thesis, Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY, 1997.

E. J. Bredensteiner and K. P. Bennett. Feature minimization within decision trees. *Computational Optimization and Applications*, 10:110–126, 1997.

L. Breiman. Bagging predictors. *Machine Learning*, 26(2):123–140, 1996.

L. Breiman. Arcing the edge. Technical Report 486, Statistics Department, University of California, June 1997.

L. Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11(7): 1493–1518, 1999.

A. Buhot and M. B. Gordon. Statistical mechanics of support vector machines. In *ESANN'99 – European Symposium on Artificial Neural Networks Proceedings*, pages 201–206. Michel Verleysen, 1999.

L. A. Bunimovich. On the ergodic properties of nowhere dispersing billiards. *Commun. Math. Phys.*, 65:295–312, 1979.

L. A. Bunimovich and Ya.G. Sinai. Markov partitions for dispersed billiards. *Commun. Math. Phys.*, 78:247–280, 1980.

C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector learning machines. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 375–381, Cambridge, MA, 1997. MIT Press.

S. Canu and A. Elisseeff. Regularization, kernels and sigmoid nets. INSA, Rouen, 1999.

B. Carl. Inequalities of Bernstein-Jackson-type and the degree of compactness of operators in Banach spaces. *Annales de l'Institut Fourier*, 35(3):79–118, 1985.

B. Carl, I. Kyrezi, and A. Pajor. Metric entropy of convex hulls in Banach spaces. *Proceedings of the London Mathematical Society*, 1999. to appear.

B. Carl and I. Stephani. *Entropy, compactness, and the approximation of operators*. Cambridge University Press, Cambridge, UK, 1990.

S. Chen. *Basis Pursuit*. PhD thesis, Department of Statistics, Stanford University, November 1995.

S. Chen, D. Donoho, and M. Saunders. Atomic decomposition by basis pursuit. *Siam Journal of Scientific Computing*, 20(1):33–61, 1999.

V. Cherkassky and F. Mulier. *Learning from Data — Concepts, Theory and Methods*. John Wiley & Sons, New York, 1998.

R. R. Coifman and M. V. Wickerhauser. Entropy-based algorithms for best-basis selection. *IEEE Transactions on Information Theory*, 38:713–718, 1992.

C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273 – 297, 1995.

R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume 1. Interscience Publishers, Inc, New York, 1953.

T. M. Cover and P. E. Hart. Nearest neighbor pattern classifications. *IEEE transaction on information theory*, 13(1):21–27, 1967.

D. Cox and F. O'Sullivan. Asymptotic analysis of penalized likelihood and related estimators. *Ann. Statist.*, 18:1676–1695, 1990.

CPLEX Optimization Incorporated, Incline Village, Nevada. *Using the CPLEX Callable Library*, 1994.

N. Cristianini, C. Campbell, and J. Shawe-Taylor. Dynamically adapting kernels in support vector machines. In D. A. Cohn M. S. Kearns, S. A. Solla, editor, *Advances in Neural Information Processing Systems*, volume 11. MIT Press, Cambridge, MA, 1999. To appear.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, UK, 2000. to appear.

N. Cristianini, J. Shawe-Taylor, and P. Sykacek. Bayesian classifiers are large margin hyperplanes in a hilbert space. In J. Shavlik, editor, *Machine Learning: Proceedings of the Fifteenth International Conference*, San Francisco, CA, 1998. Morgan Kaufmann.

I. Daubechies. *Ten Lectures on Wavelets*. CBMS-NSF Regional Conferences Series in Applied Mathematics. SIAM, Philadelphia, PA, 1992. Notes from the 1990 CBMS-NSF Conference on Wavelets and Applications at Lowell, MA.

A. de Moraes and I. R. Dunsmore. Predictive comparisons in ordinal models.

Communications in Statistics – Theory and Methods, 24(8):2145–2164, 1995.

R. A. DeVore. Nonlinear approximation. *Acta Numerica*, pages 51–150, 1998.

L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Number 31 in Applications of mathematics. Springer, New York, 1996.

L. Devroye and G. Lugosi. Lower bounds in pattern recognition and learning. *Pattern Recognition*, 28, 1995.

J. Diestel, H. Jarchow, and A. Tonge. *Absolutely Summing Operators*. Cambridge University Press, Cambridge, 1995.

R. Dietrich, M. Opper, and H. Sompolinsky. Statistical mechanics of support vector networks. *Physical Review Letters*, 82(14):2975–2978, 1999.

T. G. Dietterich. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10:1895–1924, 1998.

S. P. Dirkse and M. C. Ferris. The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems. *Optimization Methods and Software*, 5:123–156, 1995. ftp://ftp.cs.wisc.edu/tech-reports/reports/93/tr1179.ps.

H. Drucker. Improving regressors using boosting techniques. In *Proc. 14th International Conference on Machine Learning*, pages 107–115. Morgan Kaufmann, 1997.

H. Drucker and C. Cortes. Boosting decision trees. In *Advances in Neural Information Processing Systems 8*, pages 479–485, 1996.

H. Drucker, R. Schapire, and P. Simard. Boosting performance in neural networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 7:705 – 719, 1993.

R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.

R. M. Dudley. A course on empirical processes. *Lecture Notes in Mathematics*, 1097:2–142, 1984.

R. M. Dudley, E. Gine, and J. Zinn. Uniform and universal Glivenko–Cantelli classes. *Journal of Theoretical Probability*, 4:485–510, 1991.

N. Duffy and D. Helmbold. A geometric approach to leveraging weak learners. In *Computational Learning Theory: 4th European Conference*, volume 1572 of *Lecture Notes in Computer Science*, pages 18–33. Springer, 1999.

S. Dumais. Using SVMs for text categorization. *IEEE Intelligent Systems*, 13(4), 1998. In: M. A. Hearst, B. Schölkopf, S. Dumais, E. Osuna, and J. Platt: Trends and Controversies — Support Vector Machines.

N. Dunford and J. T. Schwartz. *Linear Operators Part II: Spectral Theory, Self Adjoint Operators in Hilbert Space*. Number VII in Pure and Applied Mathematics. John Wiley & Sons, New York, 1963.

N. Dunkin, J. Shawe-Taylor, and P. Koiran. A new incremental learning technique. In *Neural Nets, WIRN Vietri-96, Proceedings of the 8th Italian Workshop on*

Neural Nets, 1997.

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic models of proteins and nucleic acids*. Cambridge University Press, 1998.

N. Dyn. Interpolation and approximation by radial and related functions. In C. K. Chui, L. L. Schumaker, and D. J. Ward, editors, *Approximation Theory, VI*, pages 211–234. Academic Press, New York, 1991.

N. Dyn, D. Levin, and S. Rippa. Numerical procedures for surface fitting of scattered data by radial functions. *SIAM J. Sci. Stat. Comput.*, 7(2):639–659, April 1986.

B. Efron and R. Tibshirani. Improvements on cross-validation: the .632+ bootstrap method. *J. Amer. Statist. Assoc*, 92:548–560, 1997.

A. Ehrenfeucht, D. Haussler, M. Kearns, and L. Valiant. A general lower bound on the number of examples needed for learning. *Information and Computation*, 82: 247–261, 1989.

T. Evgeniou and M. Pontil. A note on the generalization performance of kernel classifiers with margin. AI memo, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1999a.

T. Evgeniou and M. Pontil. On the V-γ dimension for regression in reproducing kernel Hilbert spaces. AI Memo 1656, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1999b.

L. Fahrmeir and G. Tutz. *Multivariate Statistical Modelling Based on Generalized Linear Models*. Springer–Verlag, 1994.

M. C. Ferris and T. S. Munson. Interfaces to PATH 3.0: Design, implementation and usage. *Computational Optimization and Applications*, 12:207–227, 1999. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/97-12.ps.

P. C. Fishburn. *Interval Orders and Interval Graphs*. Jon Wiley and Sons, 1985.

M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

M. Frean and T. Downs. A simple cost function for boosting. Technical report, Dept. of Computer Science and Electrical Engineering, University of Queensland, 1998.

Y. Freund. An adaptive version of the boost by majority algorithm. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, 1999. (to appear).

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.

Y. Freund and R. 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, August 1997.

Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. In J. Shavlik, editor, *Machine Learning: Proceedings of the Fifteenth International Conference*, San Francisco, CA, 1998. Morgan Kaufmann.

J. Friedman. Greedy function approximation: a gradient boosting machine. Technical report, Stanford University, 1999.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. Technical report, Stanford University, 1998.

T. Frieß. Support vector neural networks: The kernel adatron with bias and soft-margin. Technical report, The University of Sheffield, 1999.

T.-T. Frieß, N. Cristianini, and C. Campbell. The kernel adatron algorithm: A fast and simple learning procedure for support vector machines. In *15th Intl. Conf. Machine Learning*. Morgan Kaufmann Publishers, 1998.

E. Gardner. The space of interactions in neural networks. *Journal of Physics A*, 21:257–70, 1988.

E. G. Gilbert. Minimizing the quadratic form on a convex set. *SIAM J. Control*, 4:61–79, 1966.

P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, 1981.

F. Girosi. Models of noise and robust estimates. AI Memo 1287, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1991.

F. Girosi. An equivalence between sparse approximation and support vector machines. *Neural Computation*, 10(6):1455–1480, 1998.

F. Girosi, M. Jones, and T. Poggio. Priors, stabilizers and basis functions: From regularization to radial, tensor and additive splines. AI Memo No. 1430, MIT, 1993.

F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995.

F. Girosi, T. Poggio, and B. Caprile. Extensions of a theory of networks for approximation and learning: outliers and negative examples. In R. Lippmann, J. Moody, and D. Touretzky, editors, *Advances in Neural Information Processing Systems 3*, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

H. Goldstein. *Classical Mechanics*. Addison-Wesley, Reading, MA, 1986.

R. P. Gorman and T. J. Sejnowsky. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1, 1988.

T. Graepel, R. Herbrich, B .Schölkopf, A. Smola, P. L. Bartlett, K.-R. Müller, K. Obermayer, and R. Williamson. Classification on proximity data with LP-machines. In D. Willshaw and A. Murray, editors, *Proceedings of ICANN'99*, volume 1, pages 304–309. IEE Press, 1999.

A. Grove and D. Schuurmans. Boosting in the limit: Maximizing the margin

of learned ensembles. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 692–699, 1998.

Y. G. Guo, P. L. Bartlett, J. Shawe-Taylor, and R. C. Williamson. Covering numbers for support vector machines. In *Proceedings of COLT99*, 1999.

L. Gurvits. A note on a scale-sensitive dimension of linear bounded functionals in banach spaces. In *Proceedings of Algorithm Learning Theory, ALT-97*, pages 352–363. Springer Verlag, 1997.

L. Gurvits and P. Koiran. Approximation and learning of convex superpositions. *Journal of Computer and System Sciences*, 55(1):161–170, 1997.

I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large VC-dimension classifiers. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 147–155. Morgan Kaufmann, San Mateo, CA, 1993.

I. Guyon, N. Matić, and V. Vapnik. Discovering informative patterns and data cleaning. In U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smythand R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 181 – 203. MIT Press, Cambridge, MA, 1996.

F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel. *Robust statistics*. Wiley, New York, NY, 1986.

G. F. Harpur and R. W. Prager. Development of low entropy coding in a recurrent network. *Network*, 7:277–284, 1996.

T. Hastie and R. Tibshirani. Classification by pairwise coupling. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

D. Haussler. Decision theoretic generalizations of the PAC model for neural net and other learning applications. *Information and Computation*, 100(1):78–150, 1992.

D. Haussler. Convolutional kernels on discrete structures. Technical Report UCSC-CRL-99-10, Computer Science Department, University of California at Santa Cruz, 1999.

S. Haykin. *Neural Networks : A Comprehensive Foundation*. Macmillan, New York, 1994.

R. Herbrich, T. Graepel, P. Bollmann-Sdorra, and K. Obermayer. Learning a preference relation in IR. In *Proceedings Workshop Text Categorization and Machine Learning, International Conference on Machine Learning 1998*, pages 80–84, 1998.

R. Herbrich, T. Graepel, and C. Cambell. Bayes point machines: Estimating the bayes point in kernel space. *IJCAI 99*, 1999a.

R. Herbrich, M. Keilbach, T. Graepel, P. Bollmann-Sdorra, and K. Obermayer. Neural networks in economics: Background, applications, and new developments. *Advances in Computational Economics*, 11:169–196, 1999b.

W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

P. J. Huber. *Robust Statistics*. John Wiley and Sons, New York, 1981.

IBM Corporation. IBM optimization subroutine library guide and reference. *IBM Systems Journal*, 31, 1992. SC23-0519.

V. V. Ivanov. *The Theory of Approximate Methods and Their Application to the Numerical Solution of Singular Integral Equations*. Nordhoff International, Leyden, 1976.

T. S. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, Cambridge, MA, 1999a. MIT Press.

T. S. Jaakkola and D. Haussler. Probabilistic kernel regression models. In *Proceedings of the 1999 Conference on AI and Statistics*, 1999b.

T. Joachims. Text categorization with support vector machines. In *European Conference on Machine Learning (ECML)*, 1998.

T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

W. Karush. Minima of functions of several variables with inequalities as side constraints. Master's thesis, Dept. of Mathematics, Univ. of Chicago, 1939.

L. Kaufman. Solving the quadratic programming problem arising in support vector classification. In B. Schölkopf, C. J. C. Burges, and A. Smola, editors, *Advances in Kernel Methods*, pages 147–168, Cambridge, MA, 1998. The MIT Press.

L. Kaufmann. Solving the quadratic programming problem arising in support vector classification. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 147–168, Cambridge, MA, 1999. MIT Press.

M. Kearns. A bound on the error of cross validation using the approximation and estimation rates, with consequences for the training-test split. *Neural Computation*, 9(5):1143–1161, 1997.

M. Kearns, Y. Mansour, A. Y. Ng, and D. Ron. An experimental and theoretical comparison of model selection methods. *Machine Learning*, 27:7–50, 1997.

M. J. Kearns and R. E. Schapire. Efficient distribution–free learning of probabilistic concepts. *Journal of Computer and System Sciences*, 48(3):464–497, 1994.

R. W. Keener and D. M. Waldman. Maximum likelihood regression of rank-censored data. *Journal of the American Statistical Association*, 80:385–392, 1985.

S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design.

Technical Report Technical Report TR-ISL-99-03, Indian Institute of Science, Bangalore, 1999. http://guppy.mpe.nus.edu.sg/~mpessk/npa_tr.ps.gz.

L. G. Khachiyan and M. J. Todd. On the complexity of approximating the maximal inscribed ellipsoid for a polytope. *Mathematical Programming*, 61:137–159, 1993.

G. Kimeldorf and G. Wahba. A correspondence between Bayesian estimation of stochastic processes and smoothing by splines. *Ann. Math. Statist.*, 41:495–502, 1971.

W. Krauth and M. Mézard. Learning algorithms with optimal stability in neural networks. *J. Phys. A*, 20:L745–L752, 1987.

H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proc. 2^{nd} Berkeley Symposium on Mathematical Statistics and Probabilistics*, pages 481–492, Berkeley, 1951. University of California Press.

R. Kühn and J. L. van Hemmen. Collective phenomena in neural networks. In E. Domany, J. L. van Hemmen, and K. Schulten, editors, *Physics of Neural Networks I.* Springer Verlag, New York, 1996.

P. F. Lambert. Designing pattern categorizers with extremal paradigm information. In S. Watanabe, editor, *Methodologies of Pattern Recognition*, pages 359–391, New York, NY, 1969. Academic Press.

P. R. Lampert. Designing pattern categories with extremal paradigm information. In M. S. Watanabe, editor, *Methodologies of Pattern Recognition.* Academic Press, N.Y., 1969.

J. Larsen and L. K. Hansen. Linear unlearning for cross-validation. *Advances in Computational Mathematics*, 5:269–280, 1996.

Y. LeCun, L. D. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Müller, E. Säckinger, P. Simard, and V. Vapnik. Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural Networks*, pages 261–276, 1995.

W. S. Lee, P. L. Bartlett, and R. C. Williamson. Efficient agnostic learning of neural networks with bounded fan-in. *IEEE Transactions on Information Theory*, 42 (6):2118–2132, November 1996.

X. Lin, G. Wahba, D. Xiang, F. Gao, R. Klein, and B. Klein. Smoothing spline ANOVA models for large data sets with Bernoulli observations and the randomized GACV. Technical Report 998, Department of Statistics, University of Wisconsin, Madison WI, 1998.

P. M. Long. The complexity of learning according to two models of a drifting environment. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pages 116–125. ACM Press, 1998.

G. G. Lorentz. *Approximation of Functions.* Chelsea Publishing Co., New York, 1986.

D. G. Luenberger. *Introduction to Linear and Nonlinear Programming.* Addison-Wesley, Reading, MA, 1973.

A. Luntz and V. Brailovsky. On estimation of characters obtained in statistical procedure of recognition (in Russian). *Technicheskaya Kibernetica*, 3, 1969.

D. J. C. MacKay. Bayesian interpolation. *Neural Computation*, 4:415–447, 1992.

D. J. C. MacKay. Introduction to Gaussian processes. available at http://wol.ra.phy.cam.ac.uk/mackay, 1997.

R. Maclin and D. Opitz. An empirical evaluation of bagging and boosting. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 546–551, 1997.

W. R. Madych and S. A. Nelson. Polyharmonic cardinal splines: a minimization property. *Journal of Approximation Theory*, 63:303–320, 1990a.

S. Mallat and Z. Zhang. Matching Pursuit in a time-frequency dictionary. *IEEE Transactions on Signal Processing*, 41:3397–3415, 1993.

O. L. Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Operations Research*, 13:444–452, 1965.

O. L. Mangasarian. Multi-surface method of pattern separation. *IEEE Transactions on Information Theory*, IT-14:801–807, 1968.

O. L. Mangasarian. *Nonlinear Programming*. SIAM, Philadelphia, PA, 1994.

O. L. Mangasarian. Mathematical programming in data mining. *Data Mining and Knowledge Discovery*, 42(1):183–201, 1997.

O. L. Mangasarian and D. R. Musicant. Successive overrelaxation for support vector machines. Technical report, University of Wisconsin, Madison, WI, USA, 1998. Tech. Report.

M. Marchand, M. Golea, and P. Ruján. Convergence theorem for sequential learning in two layer perceptrons. *Europhysics Letters*, 11:487–492, 1989.

J. L. Marroquin, S. Mitter, and T. Poggio. Probabilistic solution of ill-posed problems in computational vision. *J. Amer. Stat. Assoc.*, 82:76–89, 1987.

L. Mason, P. L. Bartlett, and J. Baxter. Improved generalization through explicit optimization of margins. *Machine Learning*, 1999. (to appear – extended abstract in NIPS 98).

MATLAB. *User's Guide*. The MathWorks, Inc., Natick, MA 01760, 1992.

B. Maurey. In: "Remarques sur un resultat non publiè de B. Maurey" by G. Pisier. In Centre de Mathematique, editor, *Seminarie d'analyse fonctionelle 1980–1981*, Palaiseau, 1981.

P. McCullagh. Regression models for ordinal data (with discussion). *Journal of the Royal Statistical Society – Series B*, 42:109–142, 1980.

P. McCullagh and J. A. Nelder. *Generalized Linear Models*. Chapman & Hall, London, 1983.

J. Mercer. Functions of positive and negative type and their connection with the theory of integral equations. *Philos. Trans. Roy. Soc. London*, A 209:415–446, 1909.

C. J. Merz and P. M. Murphy. UCI repository of machine learning databases, 1998. [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science.

M. Mézard, G. Parisi, and M. G. Virasoro. *Spin Glass Theory and Beyond*. World Scientific, Singapore, 1987.

B. F. Michell, V. F. Demyanov, and V. N. Malozemov. Finding the point of polyhedron closest to the origin. *SIAM J. Control*, 12:19–26, 1974.

F. Mosteller and R. Rourke. *Sturdy Statistics*. Addison-Wesley, Reading, MA, 1973.

B. A. Murtagh and M. A. Saunders. Minos 5.5 user's guide(rev). Technical Report SOL 83-20R, Department of Operation Research, Stanford University, Stanford CA, 1998.

R. Neal. *Bayesian Learning in Neural Networks*. Springer Verlag, 1996.

R. M. Neal. Monte carlo implementation of gaussian process models for bayesian regression and classification. Technical Report Technical Report 9702, Dept. of Statistics, 1997.

P. Niyogi and F. Girosi. On the relationship between generalization error, hypothesis complexity, and sample complexity for radial basis functions. *Neural Computation*, 8:819–842, 1996.

A. B. J. Novikov. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622. Polytechnic Institute of Brooklyn, 1962.

E. Oja. *Subspace methods of pattern recognition*. John Wiley, New York, NY, 1983.

B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381:607–609, 1996.

M. Opper. Learning in neural networks: Solvable dynamics. *Europhysics Letters*, 8 (4):389–392, 1989.

M. Opper. On the annealed VC entropy for margin classifiers: A statistical mechanics study. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in kernel methods – SV Machines*, pages 117–126. MIT Press, Cambridge MA, 1999.

M. Opper and D. Haussler. Generalization performance of bayes optimal classification algorithm for learning a perceptron. *Physical Review Letters*, 66:2677, 1991.

M. Opper and W. Kinzel. Physics of generalization. In E. Domany, J. L. van Hemmen, and K. Schulten, editors, *Physics of Neural Networks III*. Springer Verlag, New York, 1996.

M. Opper and O. Winther. Gaussian processes for classification. Submitted to Neural Computation, 1999a.

M. Opper and O. Winther. Mean field methods for classification with gaussian processes. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in*

Neural Information Processing Systems 11, Cambridge, MA, 1999b. MIT Press.

M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio. Pedestrian detection using wavelet templates. In *Proc. Computer Vision and Pattern Recognition*, pages 193–199, Puerto Rico, June 16–20 1997.

E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop*, pages 276 – 285, New York, 1997a. IEEE.

E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proc. Computer Vision and Pattern Recognition '97*, pages 130–136, 1997b.

G. Parisi. *Statistical Field Theory*. Addison-Wesley, Reading, Massachusetts, 1988.

E. Parzen. An approach to time series analysis. *Ann. Math. Statist.*, 32:951–989, 1962a.

E. Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33(3):1065–1076, 1962b.

P. J. Phillips. Support Vector Machines applied to face recognition. In *Proceedings of the Neural Information Processing Conference*, Denver, USA, 1999.

J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

T. Poggio. On optimal nonlinear associative recall. *Biological Cybernetics*, 19: 201–209, 1975.

T. Poggio and F. Girosi. A theory of networks for approximation and learning. Technical Report AIM-1140, Artificial Intelligence Laboratory, Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts, July 1989.

T. Poggio and F. Girosi. Networks for Approximation and Learning. In C. Lau, editor, *Foundations of Neural Networks*, pages 91–106. IEEE Press, Piscataway, NJ, 1992.

T. Poggio and F. Girosi. A sparse representation for function approximation. *Neural Computation*, 10:1445–1454, 1998.

T. Poggio, V. Torre, and C. Koch. Computational vision and regularization theory. *Nature*, 317:314–319, 1985.

D. Pollard. *Convergence of stochastic processes*. Springer-Verlag, Berlin, 1984.

M. Pontil, S. Mukherjee, and F. Girosi. On the noise model of support vector machine regression. AI Memo 1651, Massachusetts Institute of Technology, 1998a.

M. Pontil, R. Rifkin, and T. Evgeniou. From regression to classification in support vector machines. AI Memo 1649, Massachusetts Institute of Technology, 1998b.

M. J. D. Powell. The theory of radial basis functions approximation in 1990. In

W. A. Light, editor, *Advances in Numerical Analysis Volume II: Wavelets, Subdivision Algorithms and Radial Basis Functions*, pages 105–210. Oxford University Press, 1992.

W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*. Cambridge University Press, Cambridge, 1992.

J. R. Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 725–730, Menlo Park, 1996a. AAAI Press / MIT Press.

J. R. Quinlan. Boosting first-order learning (invited lecture). *Lecture Notes in Computer Science*, 1160:143, 1996b.

G. Rätsch. Ensemble learning for classification. Master's thesis, University of Potsdam, 1998. in German.

G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. Technical Report NC-TR-1998-021, Department of Computer Science, Royal Holloway, University of London, Egham, UK, 1998. Submitted to Machine Learning.

G. Ridgeway, D. Madigan, and T. Richardson. Boosting methodology for regression problems. In *Proc. 15th International Conference on Machine Learning*. Morgan Kaufmann, 1998.

R. Rifkin, M. Pontil, and A. Verri. A note on support vector machine degeneracy. AI Memo 1661, Massachusetts Institute of Technology, 1999.

B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, 1996.

J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

P. Ruján. Playing billiard in version space. *Neural Computation*, 9:99–122, 1997.

P. Ruján and M. Marchand. Learning by minimizing resources in neural networks. *Complex Systems*, 3:229–242, 1989.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. Rumelhart and J. McClelland, editors, *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, Cambridge, MA, 1986a.

D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, volume 1. MIT Press, Cambridge, MA, 1986b.

S. Saitoh. *Theory of Reproducing Kernels and its Applications*. Longman Scientific & Technical, Harlow, England, 1988.

G. Salton. *Automatic Information Organization and Retrieval*. McGraw–Hill, New

York, 1968.

C. Saunders, M. O. Stitson, J. Weston, L. Bottou, B. Schölkopf, and A. Smola. Support vector machine - reference manual. Technical Report CSD-TR-98-03, Department of Computer Science, Royal Holloway, University of London, Egham, TW20 0EX, UK, 1998. TR available as http://www.dcs.rhbnc.ac.uk/research/compint/areas/comp_learn/sv/pub/report98-03.ps; SVM available at http://svm.dcs.rhbnc.ac.uk/.

R. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

R. Schapire, Y. Freund, P. L. Bartlett, and W. Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 1998. (An earlier version appeared in: D. H. Fisher, Jr. (ed.), Proceedings ICML97, Morgan Kaufmann.).

R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. In *Proc. 11th Annual Conference on Computational Learning Theory*, pages 80–91, New York, NY, 1998. ACM Press.

B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings, First International Conference on Knowledge Discovery & Data Mining*. AAAI Press, Menlo Park, CA, 1995.

B. Schölkopf. *Support Vector Learning*. R. Oldenbourg Verlag, Munich, 1997.

B. Schölkopf, C. J. C. Burges, and A. J. Smola. *Advances in Kernel Methods — Support Vector Learning*. MIT Press, Cambridge, MA, 1999a.

B. Schölkopf, S. Mika, C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. Smola. Input space vs. feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5):1000 – 1017, 1999b.

B. Schölkopf, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Kernel-dependent support vector error bounds. In D. Willshaw and A. Murray, editors, *Proceedings of ICANN'99*, volume 1, pages 103–108. IEE Press, 1999.

B. Schölkopf, P. Simard, A. Smola, and V. Vapnik. Prior knowledge in support vector kernels. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 640 – 646, Cambridge, MA, 1998a. MIT Press.

B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998b.

B. Schölkopf, A. Smola, R. Williamson, and P. L. Bartlett. New support vector algorithms. NeuroCOLT Technical Report NC-TR-98-031, Royal Holloway College, University of London, UK, 1998c. To appear in *Neural Computation*.

B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2000. to appear.

B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik.

Comparing support vector machines with gaussian kernels to radial basis function classifiers. *IEEE Trans. Sign. Processing*, 45:2758 – 2765, 1997.

B. Schölkopf, R.C. Williamson, A.J. Smola, and J. Shawe-Taylor. SV estimation of a distribution's support. Accepted for NIPS'99, 1999.

L. L. Schumaker. *Spline functions: basic theory.* John Wiley and Sons, New York, 1981.

J. Schürmann. *Pattern Classification.* Wiley Interscience, New York, NY, 1996.

H. Schwenk and Y. Bengio. Training methods for adaptive boosting of neural networks. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

H. S. Seung, H. Sompolinsky, and N. Tishby. Statistical mechanics of learning from examples. *Physical Review A*, 45(8):6056–6091, 1992.

J. Shawe-Taylor, P. L. Bartlett, R. C. Williamson, and M. Anthony. Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940, 1998.

J. Shawe-Taylor and N. Cristianini. Robust bounds on the generalization from the margin distribution. NeuroCOLT Technical Report NC-TR-98-029, Royal Holloway College, University of London, UK, 1998.

J. Shawe-Taylor and N. Cristianini. Further results on the margin distribution. In *Proceedings of the Twelth Annual Conference on Computational Learning Theory, COLT'99*, 1999a.

J. Shawe-Taylor and N. Cristianini. Margin distribution bounds on generalization. In *Proceedings of the European Conference on Computational Learning Theory, EuroCOLT'99*, pages 263–273, 1999b.

P. Simard, Y. LeCun, and J. Denker. Efficient pattern recognition using a new transformation distance. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 50–58, San Mateo, CA, 1993. Morgan Kaufmann.

H. U. Simon. General bounds on the number of examples needed for learning probabilistic concepts. *J. of Comput. Syst. Sci.*, 52(2):239–254, 1996. Earlier version in 6th COLT, 1993.

A. Smola and B. Schölkopf. From regularization operators to support vector kernels. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems 10*, pages 343 – 349, Cambridge, MA, 1998a. MIT Press.

A. Smola and B. Schölkopf. On a kernel-based method for pattern recognition, regression, approximation and operator inversion. *Algorithmica*, 22:211 – 231, 1998b.

A. Smola, B. Schölkopf, and K.-R. Müller. The connection between regularization operators and support vector kernels. *Neural Networks*, 11:637–649, 1998a.

A. Smola, B. Schölkopf, and K.-R. Müller. General cost functions for support vector regression. In T. Downs, M. Frean, and M. Gallagher, editors, *Proc. of the Ninth Australian Conf. on Neural Networks*, pages 79 – 83, Brisbane, Australia, 1998b. University of Queensland.

A. Smola, R. C. Williamson, S. Mika, and B. Schölkopf. Regularized principal manifolds. In *Computational Learning Theory: 4th European Conference*, volume 1572 of *Lecture Notes in Artificial Intelligence*, pages 214 – 229. Springer, 1999.

A. J. Smola. *Learning with Kernels*. PhD thesis, Technische Universität Berlin, 1998.

A. J. Smola and B. Schölkopf. A tutorial on support vector regression. NeuroCOLT Technical Report NC-TR-98-030, Royal Holloway College, University of London, UK, 1998.

M. Sobel. Bayes and empirical Bayes procedures for comparing parameters. *Journal of the American Statistical Association*, 88:687–693, 1993.

B. Ster and A. Dobnikar. Neural networks in medical diagnosis: Comparison with other methods. In A. Bulsari et. al., editor, *Proceedings of the International Conference EANN'96*, pages 427–430, 1996.

P. Suppes, D. H. Krantz, R. D. Luce, and A. Tversky. *Foundations of Measurement Vol. II*. Academic Press Inc., San Diego, 1989.

M. Talagrand. New Gaussian estimates for enlarged balls. *Geometric and Functional Analysis*, 3(5):502–526, 1993.

M. Talagrand. Sharper bounds for gaussian and empirical processes. *Annals of Probability*, 22:28–76, 1994.

A. Tangian and J. Gruber. Constructing quadratic and polynomial objective functions. In *Proceedings of the 3rd International Conference on Econometric Decision Models*, pages 166–194, Schwerte, Germany, 1995. Springer.

A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-posed Problems*. W. H. Winston, Washington, D.C., 1977.

L. G. Valiant. A theory of learnable. *Proc. of the 1984 STOC*, pages 436–445, 1984.

R. Vanderbei. LOQO: An interior point code for quadratic programming. Technical Report SOR 94-15, Princeton University, 1994.

P. Vannerem, K.-R. Müller, A.J. Smola, B. Schölkopf, and S. Söldner-Rembold. Classifying lep data with support vector algorithms. In *Proceedings of AI-HENP'99*, 1999.

V. Vapnik. *Estimation of Dependences Based on Empirical Data [in Russian]*. Nauka, Moscow, 1979. (English translation: Springer Verlag, New York, 1982).

V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.

V. Vapnik. *Statistical Learning Theory*. Wiley, New York, 1998.

V. Vapnik and A. Chervonenkis. On the uniform convergence of relative frequencies

of events to their probabilities. *Theory of Probability and its Applications*, 16(2): 264–280, 1971.

V. Vapnik and A. Chervonenkis. Necessary and sufficient conditions for the uniform convergence of means to their expectations. *Theory of Probability and its Applications*, 26(3):532–553, 1981.

V. Vapnik and A. Chervonenkis. The necessary and sufficient conditions for consistency in the empirical risk minimization method. *Pattern Recognition and Image Analysis*, 1(3):283–305, 1991.

V. Vapnik, S. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 281–287, Cambridge, MA, 1997. MIT Press.

V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 1963.

V. Vovk, C. Saunders, and A. Gammermann. Ridge regression learning algorithm in dual variables. In J. Shavlik, editor, *Machine Learning Proceedings of the Fifteenth International Conference(ICML '98)*, San Francisco, CA, 1998. Morgan Kaufmann.

G. Wahba. Convergence rates of certain approximate solutions to Fredholm integral equations of the first kind. *Journal of Approximation Theory*, 7:167 – 185, 1973.

G. Wahba. Spline bases, regularization, and generalized cross-validation for solving approximation problems with large quantities of noisy data. In J. Ward and E. Cheney, editors, *Proceedings of the International Conference on Approximation theory in honour of George Lorenz*, pages 8–10, Austin, TX, 1980. Academic Press.

G. Wahba. Constrained regularization for ill posed linear operator equations, with applications in meteorology and medicine. In S. Gupta and J. Berger, editors, *Statistical Decision Theory and Related Topics, III, Vol.2*, pages 383–418. Academic Press, 1982.

G. Wahba. A comparison of GCV and GML for choosing the smoothing parameter in the generalized spline smoothing problem. *Ann. Statist.*, 13:1378–1402, 1985.

G. Wahba. *Spline Models for Observational Data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, 1990.

G. Wahba. Multivariate function and operator estimation, based on smoothing splines and reproducing kernels. In M. Casdagli and S. Eubank, editors, *Nonlinear Modeling and Forecasting, SFI Studies in the Sciences of Complexity, Proc. Vol XII*, pages 95–112. Addison-Wesley, 1992.

G. Wahba. The bias-variance tradeoff and the randomized GACV. In D. A. Cohn M. S. Kearns, S. A. Solla, editor, *Advances in Neural Information Processing Systems*, volume 11. MIT Press, Cambridge, MA, 1999a. To appear.

G. Wahba. Support vector machines, reproducing kernel hilbert spaces and the

randomized GACV. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 69–88, Cambridge, MA, 1999b. MIT Press.

T. Watkin. Optimal learning with a neural network. *Europhysics Letters*, 21:871, 1993.

T. L. H. Watkin, A. Rau, and M. Biehl. The statistical mechanics of learning a rule. *Reviews of Modern Physics*, 65:499–556, 1993.

J. Weston. Leave–one–out support vector machines. In *Proceedings of the International Joint Conference on Artifical Intelligence*, Sweden, 1999.

J. Weston, A Gammerman, M. O. Stitson, V. Vapnik, V. Vovk, and C. Watkins. Density estimation using SV machines. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — SV Learning*, Cambridge, MA, 1999. MIT Press.

J. Weston and C. Watkins. Multi-class support vector machines. Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, Egham, TW20 0EX, UK, 1998.

C. K. I. Williams. Computation with infinite networks. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 295–301, Cambridge, MA, 1997. MIT Press.

C. K. I. Williams. Prediction with gaussian processes: From linear regression to linear prediction and beyond. In M. I. Jordan, editor, *Learning and Inference in Graphical Models*. Kluwer, 1998.

R. Williamson, A. Smola, and B. Schölkopf. Entropy numbers, operators and support vector kernels. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 127–144, Cambridge, MA, 1999. MIT Press.

R. C. Williamson, A. J. Smola, and B. Schölkopf. Generalization performance of regularization networks and support vector machines via entropy numbers of compact operators. NeuroCOLT Technical Report NC-TR-98-019, Royal Holloway College, University of London, UK, 1998.

P. Wolfe. A duality theorem for nonlinear programming. *Quarterly of Applied Mathematics*, 19:239–244, 1961.

S. K. M. Wong, Y. Y. Yao, and P. Bollmann. Linear structure in information retrieval. In *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 219–232, 1988.

J. Ye and W. Wong. Evaluation of highly complex modeling procedures with Binomial and Poisson data. Manuscript, University of Chicago School of Business, 1997.

A. N. Zemlyakov and A. .B. Katok. The topological transitivity of billiards in polygons. *mat. zametki*, 18:291–301, 1975.

Index