An Introduction to Support Vector Machines and Other Kernel-based Learning Methods

by Nello Cristianini and John Shawe-Taylor ISBN:0521780195

Cambridge University Press ?2000 (190 pages)

This is the first comprehensive introduction to SVMs, a new generation learning system based on recent advances in statistical learning theory; it will help readers understand the theory and its real-world applications.

Companion Web Site

Table of Contents

An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods

Nello Cristianini
John Shawe-Taylor
CAMBRIDGE UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011-4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa

http://www.cambridge.org

First published 2000
Reprinted 2000 (with corrections), 2001 (twice), 2002 (with corrections), 2003

Typeface Times 10/12pt. *System* LATEX 2 [EPC]

A catalogue record for this book is available from the British Library

Library of Congress Cataloguing in Publication data available

0521780195

An Introduction to Support Vector Machines

This book is the first comprehensive introduction to Support Vector Machines (SVMs), a new generation learning system based on recent advances in statistical learning theory. SVMs deliver state-of-the-art performance in real-world applications such as text categorisation, hand-written character recognition, image classification, biosequence analysis, etc.

Their first introduction in the early '90s led to an explosion of applications and deepening theoretical analysis, that has now established Support Vector Machines as one of the standard tools for machine learning and data mining. Students will find the book both stimulating and accessible, while practitioners will be guided smoothly through the material required for a good grasp of the theory and application of these techniques. The concepts are introduced gradually in accessible and self-contained stages, while in each stage the presentation is rigorous and thorough. Pointers to relevant literature and web sites containing software ensure that it forms an ideal starting point for further study. Equally the book will equip the practitioner to apply the techniques and its associated web site will provide pointers to updated literature, new applications, and on-line software.

Nello Cristianini was born in Gorizia, Italy. He has studied at University of Trieste in Italy; Royal Holloway, University of London; the University of Bristol; and the University of California in Santa Cruz. He is an active young researcher in the theory and applications of Support Vector Machines and other learning systems and has published in a number of key international conferences and journals in this area.

John Shawe-Taylor was born in Cheltenham, England. He studied at the University of Cambridge; University of Ljubljana in Slovenia; Simon Fraser University in Canada; Imperial College; and Royal Holloway, University of London. He has published widely on the theoretical analysis of learning systems in addition to other areas of discrete mathematics and computer science. He is a professor of Computing Science at Royal Holloway, University of London. He is currently the co-ordinator of a European funded collaboration of sixteen universities involved in research on Neural and Computational Learning.

Preface

In the last few years there have been very significant developments in the theoretical understanding of Support Vector Machines (SVMs) as well as algorithmic strategies for implementing them, and applications of the approach to practical problems. We believe that the topic has reached the point at which it should perhaps be viewed as its own subfield of machine learning, a subfield which promises much in both theoretical insights and practical usefulness. Despite reaching this stage of development, we were aware that no organic integrated introduction to the subject had yet been attempted. Presenting a comprehensive introduction to SVMs requires the synthesis of a surprisingly wide range of material, including dual representations, feature spaces, learning theory, optimisation theory, and algorithmics. Though active research is still being pursued in all of these areas, there are stable foundations in each that together form the basis for the SVM concept. By building from those stable foundations, this book attempts a measured and accessible introduction to the subject of Support Vector Machines.

The book is intended for machine learning students and practitioners who want a gentle but rigorous introduction to this new class of learning systems. It is organised as a textbook that can be used either as a central text for a course on SVMs, or as an additional text in a neural networks, machine learning, or pattern recognition class. Despite its organisation as a textbook, we have kept the presentation self-contained to ensure that it is suitable for the interested scientific reader not necessarily working directly in machine learning of computer science. In this way the book should give readers from other scientific disciplines a practical introduction to Support Vector Machines enabling them to apply the approach to problems from their own domain. We have attempted to provide the reader with a route map through the rigorous derivation of the material. For this reason we have only included proofs or proof sketches where they are accessible and where we feel that they enhance the understanding of the main ideas. Readers who are interested in the detailed proofs of the quoted results are referred to the original articles.

Exercises are provided at the end of the chapters, as well as pointers to relevant literature and on-line software and articles. Given the potential instability of on-line material, in some cases the book points to a dedicated website, where the relevant links will be kept updated, hence ensuring that readers can continue to access on-line software and articles. We have always endeavoured to make clear who is responsible for the material even if the pointer to it is an indirect one. We hope that authors will not be offended by these occasional indirect pointers to their work. Each chapter finishes with a section entitled Further Reading and Advanced Topics, which fulfils two functions. First by moving all the references into this section we have kept the main text as uncluttered as possible. Again we ask for the indulgence of those who have contributed to this field when we quote their work but delay giving a reference until this section. Secondly, the section is intended to provide a starting point for readers who wish to delve further into the topics covered in that chapter. The references will also be held and kept up to date on the website. A further motivation for moving the references out of the main body of text is the fact that the field has now reached a stage of maturity which justifies our unified presentation. The two exceptions we have made to this rule are firstly for theorems which are generally known by the name of the original author such as Mercer's theorem, and secondly in Chapter 8 which describes specific experiments reported in the research literature.

The fundamental principle that guided the writing of the book is that it should be accessible to students and practitioners who would prefer to avoid complicated proofs and definitions on their way to using SVMs. We believe that by developing the material in intuitively appealing but rigorous stages, in fact SVMs appear as simple and natural systems. Where possible we first introduce concepts in a simple example, only then showing how they are used in more complex cases. The book is self-contained, with an appendix providing any necessary mathematical tools beyond basic linear algebra and probability. This makes it suitable for a very interdisciplinary audience.

Much of the material was presented in five hours of tutorials on SVMs and large margin generalisation held at the University of California at Santa Cruz during 1999, and most of the feedback received from these was incorporated into the book. Part of this book was written while Nello was visiting the University of California

at Santa Cruz, a wonderful place to work thanks to both his hosts and the environment of the campus. During the writing of the book, Nello made frequent and long visits to Royal Holloway, University of London. Nello would like to thank Lynda and her family for hosting him during these visits. Together with John he would also like to thank Alex Gammerman, the technical and administrative staff, and academic colleagues of the Department of Computer Science at Royal Holloway for providing a supportive and relaxed working environment, allowing them the opportunity to concentrate on the writing.

Many people have contributed to the shape and substance of the book, both indirectly through discussions and directly through comments on early versions of the manuscript. We would like to thank Kristin Bennett, Colin Campbell, Nicolo Cesa-Bianchi, David Haussler, Ralf Herbrich, Ulrich Kockelkorn, John Platt, Tomaso Poggio, Bernhard Schölkopf, Alex Smola, Chris Watkins, Manfred Warmuth, Chris Williams, and Bob Williamson.

We would also like to thank David Tranah and Cambridge University Press for being so supportive and helpful in the processing of the book. Alessio Cristianini assisted in the establishment of the website. Kostantinos Veropoulos helped to create the pictures for Chapter 6 which were generated using his software package at the University of Bristol. We would like to thank John Platt for providing the SMO pseudocode included in Appendix A.

Nello would like to thank the EPSRC for supporting his research and Colin Campbell for being a very understanding and helpful supervisor. John would like to thank the European Commission for support through the NeuroCOLT2 Working Group, EP27150.

Since the first edition appeared a small number of errors have been brought to our attention, and we have endeavoured to ensure that they were all corrected before reprinting. We would be grateful if anyone discovering further problems contact us through the feedback facility on the book's web page **http://www.support-vector.net.**

Nello Cristianini and John Shawe-Taylor
June, 2000

Notation

N	dimension of feature space
$y \in Y$	output and output space
$\mathbf{x} \quad X$	input and input space
F	feature space
	general class of real-valued functions
	class of linear functions
$\mathbf{x} \cdot \mathbf{z}$	inner product between $\mathbf{x}$ and $\mathbf{z}$
$: X \quad F$	mapping to feature space
$K(\mathbf{x},\mathbf{z})$	kernel $(\mathbf{x}) \cdot (\mathbf{z})$
$f(\mathbf{x})$	real- valued function before thresholding
n	dimension of input space
R	radius of the ball containing the data
-insensitive	loss function insensitive to errors less than
$\mathbf{w}$	weight vector
b	bias
	dual variables or Lagrange multipliers

L	primal Lagrangian
W	dual Lagrangian
$\|\cdot\|_p$	p-norm
ln	natural logarithm
e	base of the natural logarithm
log	logarithm to the base 2
$\mathbf{x}'$, $\mathbf{X}'$	transpose of vector, matrix
$\mathbb{N}$, $\mathbb{R}$	natural, real numbers
S	training sample
	training set size
	learning rate
	error probability
	confidence
	margin
	slack variables
d	VC dimension

Chapter 1: The Learning Methodology

Overview

The construction of machines capable of learning from experience has for a long time been the object of both philosophical and technical debate. The technical aspect of the debate has received an enormous impetus from the advent of electronic computers. They have demonstrated that machines can display a significant level of learning ability, though the boundaries of this ability are far from being clearly defined.

The availability of reliable learning systems is of strategic importance, as there are many tasks that cannot be solved by classical programming techniques, since no mathematical model of the problem is available. So for example it is not known how to write a computer program to perform hand-written character recognition, though there are plenty of examples available. It is therefore natural to ask if a computer could be trained to recognise the letter ‘A' from examples - after all this is the way humans learn to read. We will refer to this approach to problem solving as the learning methodology

The same reasoning applies to the problem of finding genes in a DNA sequence, filtering email, detecting or recognising objects in machine vision, and so on. Solving each of these problems has the potential to revolutionise some aspect of our life, and for each of them machine learning algorithms could provide the key to its solution.

In this chapter we will introduce the important components of the learning methodology, give an overview of the different kinds of learning and discuss why this approach has such a strategic importance. After the framework of the learning methodology has been introduced, the chapter ends with a roadmap for the rest of the book, anticipating the key themes, and indicating why Support Vector Machines meet many of the challenges confronting machine learning systems. As this roadmap will describe the role of the different chapters, we urge our readers to refer to it before delving further into the book.

2 Chapter 1: The Learning Methodology

1.1 Supervised Learning

When computers are applied to solve a practical problem it is usually the case that the method of deriving the required output from a set of inputs can be described explicitly. The task of the system designer and eventually the programmer implementing the specifications will be to translate that method into a sequence of instructions which the computer will follow to achieve the desired effect.

As computers are applied to solve more complex problems, however, situations can arise in which there is no known method for computing the desired output from a set of inputs, or where that computation may be very expensive. Examples of this type of situation might be modelling a complex chemical reaction, where the precise interactions of the different reactants are not known, or classification of protein types based on the DNA sequence from which they are generated, or the classification of credit applications into those who will default and those who will repay the loan.

These tasks cannot be solved by a traditional programming approach since the system designer cannot precisely specify the method by which the correct output can be computed from the input data. An alternative strategy for solving this type of problem is for the computer to attempt to learn the input/output functionality from examples, in the same way that children learn which are sports cars simply by being told which of a large number of cars are sporty rather than by being given a precise specification of sportiness. The approach of using examples to synthesise programs is known as the *learning methodology,* and in the particular case when the examples are input/output pairs it is called *supervised learning.* The examples of input/output functionality are referred to as the *training data.*

The input/output pairings typically reflect a functional relationship mapping inputs to outputs, though this is not always the case as for example when the outputs are corrupted by noise. When an underlying function from inputs to outputs exists it is referred to as the *target function.* The estimate of the target function which is learnt or output by the learning algorithm is known as the *solution* of the learning problem. In the case of classification this function is sometimes referred to as the *decision function.* The solution is chosen from a set of candidate functions which map from the input space to the output domain. Usually we will choose a particular set or class of candidate functions known as *hypotheses* before we begin trying to learn the correct function. For example, so-called *decision trees* are hypotheses created by constructing a binary tree with simple decision functions at the internal nodes and output values at the leaves. Hence, we can view the choice of the set of hypotheses (or *hypothesis space*) as one of the key ingredients of the learning strategy. The algorithm which takes the training data as input and selects a hypothesis from the hypothesis space is the second important ingredient. It is referred to as the *learning algorithm.*

In the case of learning to distinguish sports cars the output is a simple yes/no tag which we can think of as a binary output value. For the problem of recognising protein types, the output value will be one of a finite number of categories, while the output values when modelling a chemical reaction might be the concentrations of the reactants given as real values. A learning problem with binary outputs is referred to as a *binary classification* problem, one with a finite number of categories as *multi-class classification,* while for real-valued outputs the problem becomes known as *regression.* This book will consider all of these types of learning, though binary classification is always considered first as it is often the simplest case.

There are other types of learning that will not be considered in this book. For example *unsupervised learning* considers the case where there are no output values and the learning task is to gain some understanding of the process that generated the data. This type of learning includes density estimation, learning the support of a distribution, clustering, and so on. There are also models of learning which consider more complex interactions between a learner and their environment. Perhaps the simplest case is when the learner is allowed to query the environment about the output associated with a particular input. The study of how this affects the learner's ability to learn different tasks is known as *query learning.* Further complexities of interaction are considered in *reinforcement learning,* where the learner has a range of actions at their disposal which they can take to attempt to move towards states where they can expect high rewards. The learning methodology can

play a part in reinforcement learning if we treat the optimal action as the output of a function of the current state of the learner. There are, however, significant complications since the quality of the output can only be assessed indirectly as the consequences of an action become clear.

Another type of variation in learning models is the way in which the training data are generated and how they are presented to the learner. For example, there is a distinction made between *batch* learning in which all the data are given to the learner at the start of learning, and *on-line* learning in which the learner receives one example at a time, and gives their estimate of the output, before receiving the correct value. In on-line learning they update their current hypothesis in response to each new example and the quality of learning is assessed by the total number of mistakes made during learning.

The subject of this book is a family of techniques for learning to perform input/output mappings from labelled examples for the most part in the batch setting, that is for applying the supervised learning methodology from batch training data.

1.2 Learning and Generalisation

We discussed how the quality of an on-line learning algorithm can be assessed in terms of the number of mistakes it makes during the training phase. It is not immediately clear, however, how we can assess the quality of a hypothesis generated during batch learning. Early machine learning algorithms aimed to learn representations of simple symbolic functions that could be understood and verified by experts. Hence, the goal of learning in this paradigm was to output a hypothesis that performed the correct classification of the training data and early learning algorithms were designed to find such an accurate fit to the data. Such a hypothesis is said to be *consistent.* There are two problems with the goal of generating a verifiable consistent hypothesis.

The first is that the function we are trying to learn may not have a simple representation and hence may not be easily verified in this way. An example of this situation is the identification of genes within a DNA sequence. Certain subsequences are genes and others are not, but there is no simple way to categorise which are which.

The second problem is that frequently training data are noisy and so there is no guarantee that there is an underlying function which correctly maps the training data. The example of credit checking is clearly in this category since the decision to default may be a result of factors simply not available to the system. A second example would be the classification of web pages into categories, which again can never be an exact science.

The type of data that is of interest to machine learning practitioners is increasingly of these two types, hence rendering the proposed measure of quality difficult to implement. There is, however, a more fundamental problem with this approach in that even when we can find a hypothesis that is consistent with the training data, it may not make correct classifications of unseen data. The ability of a hypothesis to correctly classify data not in the training set is known as its *generalisation,* and it is this property that we shall aim to optimise.

Shifting our goal to generalisation removes the need to view our hypothesis as a correct representation of the true function. If the hypothesis gives the right output it satisfies the generalisation criterion, which in this sense has now become a functional measure rather than a descriptional one. In this sense the criterion places no constraints on the size or on the ‘meaning' of the hypothesis - for the time being these can be considered to be arbitrary.

This change of emphasis will be somewhat counteracted when we later search for compact representations (that is short descriptions) of hypotheses, as these can be shown to have good generalisation properties, but for the time being the change can be regarded as a move from symbolic to subsymbolic representations.

A precise definition of these concepts will be given in Chapter 4, when we will motivate the particular models we shall be using.

2 Chapter 1: The Learning Methodology

1.3 Improving Generalisation

The generalisation criterion places an altogether different constraint on the learning algorithm. This is most amply illustrated by the extreme case of rote learning. Many classical algorithms of machine learning are capable of representing any function and for difficult training sets will give a hypothesis that behaves like a rote learner. By a rote learner we mean one that correctly classifies the data in the training set, but makes essentially uncorrelated predictions on unseen data. For example, decision trees can grow so large that there is a leaf for each training example. Hypotheses that become too complex in order to become consistent are said to *overfit.* One way of trying to control this difficulty is to restrict the size of the hypothesis, for example pruning the size of the decision tree. Ockham's razor is a principle that motivates this approach, suggesting that unnecessary complications are not helpful, or perhaps more accurately complications must pay for themselves by giving significant improvements in the classification rate on the training data.

These ideas have a long history as the mention of Ockham suggests. They can be used to motivate heuristic trade-offs between complexity and accuracy and various principles have been proposed for choosing the optimal compromise between the two. As an example the *Minimum Description Length* (MDL) principle proposes to use the set of hypotheses for which the description of the chosen function together with the list of training errors is shortest.

The approach that we will adopt is to motivate the trade-off by reference to statistical bounds on the generalisation error. These bounds will typically depend on certain quantities such as the margin of the classifier, and hence motivate algorithms which optimise the particular measure. The drawback of such an approach is that the algorithm is only as good as the result that motivates it. On the other hand the strength is that the statistical result provides a well-founded basis for the approach, hence avoiding the danger of a heuristic that may be based on a misleading intuition.

The fact that the algorithm design is based on a statistical result does not mean that we ignore the computational complexity of solving the particular optimisation problem. We are interested in techniques that will scale from toy problems to large realistic datasets of hundreds of thousands of examples. It is only by performing a principled analysis of the computational complexity that we can avoid settling for heuristics that work well on small examples, but break down once larger training sets are used. The theory of computational complexity identifies two classes of problems. For the first class there exist algorithms that run in time polynomial in the size of the input, while for the second the existence of such an algorithm would imply that any problem for which we can check a solution in polynomial time can also be solved in polynomial time. This second class of problems is known as the NP-complete problems and it is generally believed that these problems cannot be solved efficiently.

In Chapter 4 we will describe in more detail the type of statistical result that will motivate our algorithms. We distinguish between results that measure generalisation performance that can be obtained with a given finite number of training examples, and asymptotic results, which study how the generalisation behaves as the number of examples tends to infinity. The results we will introduce are of the former type, an approach that was pioneered by Vapnik and Chervonenkis.

We should emphasise that alternative algorithms to those we will describe can be motivated by other approaches to analysing the learning methodology. We will point to relevant literature describing some other approaches within the text. We would, however, like to mention the Bayesian viewpoint in a little more detail at this stage.

The starting point for Bayesian analysis is a prior distribution over the set of hypotheses that describes the learner's prior belief of the likelihood of a particular hypothesis generating the data. Once such a prior has been assumed together with a model of how the data have been corrupted by noise, it is possible in principle to estimate the most likely hypothesis given the particular training set, and even to perform a weighted average over the set of likely hypotheses.

If no restriction is placed over the set of all possible hypotheses (that is all possible functions from the input space to the output domain), then learning is impossible since no amount of training data will tell us how to classify unseen examples. Problems also arise if we allow ourselves the freedom of choosing the set of hypotheses after seeing the data, since we can simply assume all of the prior probability on the correct hypotheses. In this sense it is true that all learning systems have to make some prior assumption of a Bayesian type often called the *learning bias.* We will place the approach we adopt in this context in Chapter 4.

1.4 Attractions and Drawbacks of Learning

It is not surprising that the promise of the learning methodology should be so tantalising. Firstly, the range of applications that can potentially be solved by such an approach is very large. Secondly, it appears that we can also avoid much of the laborious design and programming inherent in the traditional solution methodology, at the expense of collecting some labelled data and running an off-the-shelf algorithm for learning the input/output mapping. Finally, there is the attraction of discovering insights into the way that humans function, an attraction that so inspired early work in neural networks, occasionally to the detriment of scientific objectivity.

There are, however, many difficulties inherent in the learning methodology, difficulties that deserve careful study and analysis. One example is the choice of the class of functions from which the input/output mapping must be sought. The class must be chosen to be sufficiently rich so that the required mapping or an approximation to it can be found, but if the class is too large the complexity of learning from examples can become prohibitive, particularly when taking into account the number of examples required to make statistically reliable inferences in a large function class. Hence, learning in three-node neural networks is known to be NP-complete, while the problem of minimising the number of training errors of a thresholded linear function is also NP-hard. In view of these difficulties it is immediately apparent that there are severe restrictions on the applicability of the approach, and any suggestions of a panacea are highly misleading.

In practice these problems manifest themselves in specific learning difficulties. The first is that the learning algorithm may prove inefficient as for example in the case of local minima. The second is that the size of the output hypothesis can frequently become very large and impractical. The third problem is that if there are only a limited number of training examples too rich a hypothesis class will lead to overfitting and hence poor generalisation. The fourth problem is that frequently the learning algorthm is controlled by a large number of parameters that are often chosen by tuning heuristics, making the system difficult and unreliable to use.

Despite the drawbacks, there have been notable successes in the application of the learning methodology to problems of practical interest. Unfortunately, however, there is frequently a lack of understanding about the conditions that will render an application successful, in both algorithmic and statistical inference terms. We will see in the next section that Support Vector Machines address all of these problems.

2 Chapter 1: The Learning Methodology

1.5 Support Vector Machines for Learning

Support Vector Machines (SVM) are learning systems that use a hypothesis space of linear functions in a high dimensional feature space, trained with a learning algorithm from optimisation theory that implements a learning bias derived from statistical learning theory. This learning strategy introduced by Vapnik and co-workers is a principled and very powerful method that in the few years since its introduction has already outperformed most other systems in a wide variety of applications.

This book gives an introduction to Support Vector Machines by describing the hypothesis space and its representation in Chapters 2 and 3, the learning bias in Chapter 4, and the learning algorithm in Chapters 5 and 7. Chapter 6 is the key chapter in which all these components are brought together, while Chapter 8 gives an overview of some applications that have been made to real-world problems. The book is written in a modular fashion so that readers familiar with the material covered in a particular chapter can safely bypass that section. In particular, if the reader wishes to get a direct introduction to what an SVM is and how to implement one, they should move straight to Chapters 6 and 7.

Chapter 2 introduces linear learning machines one of the main building blocks of the system. Chapter 3 deals with kernel functions which are used to define the implicit feature space in which the linear learning machines operate. The use of kernel functions is the key to the efficient use of high dimensional feature spaces. The danger of overfitting inherent in high dimensions requires a sophisticated learning bias provided by the statistical learning theory covered in Chapter 4. Optimisation theory covered in Chapter 5 gives a precise characterisation of the properties of the solution which guide the implementation of efficient learning algorithms described in Chapter 7, and ensure that the output hypothesis has a compact representation. The particular choice of a convex learning bias also results in the absence of local minima so that solutions can always be found efficiently even for training sets with hundreds of thousands of examples, while the compact representation of the hypothesis means that evaluation on new inputs is very fast. Hence, the four problems of efficiency of training, efficiency of testing, overfitting and algorithm parameter tuning are all avoided in the SVM design.

2 Chapter 1: The Learning Methodology

1.6 Exercises

1. Describe the process of distinguishing between reptiles and mammals as a binary classification problem. What is the input space? How might the inputs be represented for computer processing of such data? Give an example of a function realising the classification rule.
2. Repeat Exercise 1 for the following categories of animals: birds/fishes; fishes/mammals; mammals/birds. Write down the list of decision functions used in the four decision rules.
3. You are given a set of correctly labelled mammals and fishes:

   {dog, cat, dolphin}, {goldfish, shark, tuna}.

   Taking the hypothesis space as the set of four functions obtained in Exercises 1 and 2, how would you pick the correct decision rule?

2 Chapter 1: The Learning Methodology

1.7 Further Reading and Advanced Topics

The problem of learning from data has been investigated by philosophers throughout history, under the name of ‘inductive inference'. Although this might seem surprising today, it was not until the 20th century that pure induction was recognised as impossible unless one assumes some prior knowledge. This conceptual achievement is essentially due to the fundamental work of Karl Popper [119].

There is a long history of studying this problem within the statistical framework. Gauss proposed the idea of least squares regression in the 18th century, while Fisher's approach [40] to classification in the 1930s still provides the starting point for most analysis and methods.

Researchers in the area of artificial intelligence started to consider the problem of learning from its very beginning. Alan Turing [154] proposed the idea of learning machines in 1950, contesting the belief of Lady Lovelace that ‘the machine can only do what we know how to order it to do'. Also there is a foresight of subsymbolic learning in that paper, when Turing comments: ‘An important feature of a learning machine is that its teacher will often be very largely ignorant of quite what is going on inside, although he may still be able to some extent to predict his pupil's behaviour.' Just a few years later the first examples of learning machines were developed, for example Arthur Samuel's draughts player [124] was an early example of reinforcement learning, while Frank Rosenblatt's perceptron [122] contained many of the features of the systems discussed in the next chapter. In particular, the idea of modelling learning problems as problems of search in a suitable hypothesis space is characteristic of the artificial intelligence approach. Solomonoff also formally studied the problem of learning as inductive inference in the famous papers [151] and [152].

The development of learning algorithms became an important sub field of artificial intelligence, eventually forming the separate subject area of *machine learning*. A very readable ‘first introduction' to many problems in machine learning is provided by Tom Mitchell's book *Machine learning* [99]. Support Vector Machines were introduced by Vapnik and his co-workers in the COLT paper [19] and is described in more detail in Vapnik's book [159].

These references are also given on the website http://www.support-vector.net, which will be kept up to date with new work, pointers to software and papers that are available on-line.

2 Chapter 1: The Learning Methodology

Chapter 2: Linear Learning Machines

In supervised learning, the learning machine is given a training set of examples (or inputs) with associated labels (or output values). Usually the examples are in the form of attribute vectors, so that the input space is a subset of $\mathbb{R}^n$. Once the attribute vectors are available, a number of sets of hypotheses could be chosen for the problem. Among these, linear functions are the best understood and simplest to apply. Traditional statistics and the classical neural networks literature have developed many methods for discriminating between two classes of instances using linear functions, as well as methods for interpolation using linear functions. These techniques, which include both efficient iterative procedures and theoretical analysis of their generalisation properties, provide the framework within which the construction of more complex systems will be developed in the coming chapters. In this chapter we review results from the literature that will be relevant to the study of Support Vector Machines. We will first discuss algorithms and issues of classification, and then we will move on to the problem of regression. Throughout this book, we will refer to learning machines using hypotheses that form linear combinations of the input variables as linear learning machines.

Importantly, we will show that in most cases such machines can be represented in a particularly useful form, which we will call the *dual representation*. This fact will prove crucial in later chapters. The important notions of *margin* and *margin distribution* are also introduced in this chapter. The classification results are all introduced for the binary or two-class case, and at the end of the chapter it is shown how to generalise them to multiple classes.

2.1 Linear Classification

Binary classification is frequently performed by using a real-valued function $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$ in the following way: the input $\mathbf{x} = (x_1, \ldots, x_n) \in X$ is assigned to the positive class, if $f(\mathbf{x}) \geq 0$, and otherwise to the negative class. We consider the case where $f(\mathbf{x})$ is a linear function of $\mathbf{x} \in X$, so that it can be written as

$$\begin{aligned} f(\mathbf{x}) &= \langle \mathbf{w} \cdot \mathbf{x} \rangle + b \\ &= \sum_{i=1}^{n} w_i x_i + b \end{aligned}$$

where $(\mathbf{w}, b) \in \mathbb{R}^n \times \mathbb{R}$ are the parameters that control the function and the decision rule is given by $\mathrm{sgn}(f(\mathbf{x}))$, where we will use the convention that $\mathrm{sgn}(0) = 1$. The learning methodology implies that these parameters must be learned from the data.

A geometric interpretation of this kind of hypothesis is that the input space X is split into two parts by the hyperplane defined by the equation $\mathbf{w} \cdot \mathbf{x} + b = 0$ (see Figure 2.1). A hyperplane is an affine subspace of dimension $n - 1$ which divides the space into two half spaces which correspond to the inputs of the two distinct classes. For example in Figure 2.1 the hyperplane is the dark line, with the positive region above and the negative region below. The vector $\mathbf{w}$ defines a direction perpendicular to the hyperplane, while varying the value of b moves the hyperplane parallel to itself. It is therefore clear that a representation involving $n + 1$ free parameters is necessary, if one wants to represent all possible hyperplanes in $\mathbb{R}^n$.

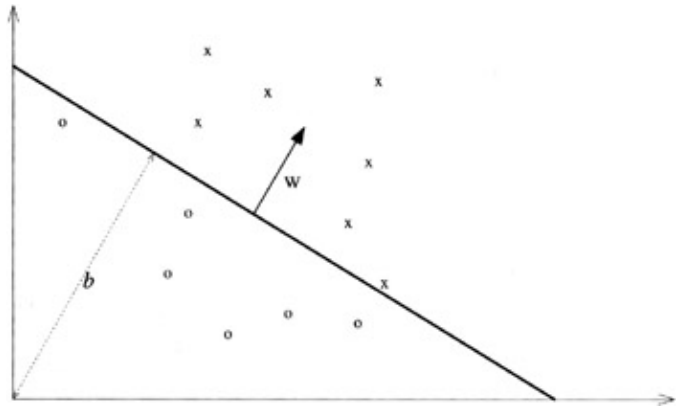


Figure 2.1: A separating hyperplane (**w**, *b*) for a two dimensional training set

Both statisticians and neural network researchers have frequently used this simple kind of classifier, calling them respectively *linear discriminants* and *perceptrons.* The theory of linear discriminants was developed by Fisher in 1936, while neural network researchers studied perceptrons in the early 1960s, mainly due to the work of Rosenblatt. We will refer to the quantities **w** and *b* as the *weight vector* and *bias,* terms borrowed from the neural networks literature. Sometimes -*b* is replaced by , a quantity known as the *threshold.*

As we are studying supervised learning from examples, we first introduce some notation we will be using throughout the book to refer to inputs, outputs, training sets, and so on.

Definition 2.1

We typically use X to denote the input space and Y to denote the output domain. Usually we will have $X \subseteq \mathbb{R}^n$, while for binary classification $Y = \{-1,1\}$, for m-class classification $Y = \{1,2,\ldots, m\}$, and for regression $Y \subseteq \mathbb{R}^n$. A *training set* is a collection of *training examples,* which are also called *training data.* It is usually denoted by

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)) \subseteq (X \times Y)^\ell ,$$

---

where is the number of examples. We refer to the x_i as *examples* or *instances* and the y_i as their *labels.* The training set S is *trivial* if the labels of all the examples are equal. Note that if X is a vector space, the input vectors are column vectors as are the weight vectors. If we wish to form a row vector from $\mathbf{x}_i$ we can take the transpose $\mathbf{x}_i'$.

Several simple iterative algorithms optimising different cost functions were introduced in the 1960s for separating points from two populations by means of a hyperplane. In the following subsections we will review some of the best-known, and will highlight some of their most interesting properties. The case of the perceptron is particularly interesting not only for historical reasons, but also because, even in such a simple system, we can already find most of the central concepts that we will need for the theory of Support Vector Machines. Note that some algorithms, such as least squares, have been used both for regression and for classification. In order to avoid duplication, we will describe them in the regression section.

2.1.1 Rosenblatt's Perceptron

The first iterative algorithm for learning linear classifications is the procedure proposed by Frank Rosenblatt in 1956 for the perceptron. The algorithm created a great deal of interest when it was first introduced. It is an 'on-line' and 'mistake-driven' procedure, which starts with an initial weight vector $\mathbf{w}_0$ (usually $\mathbf{w}_0 = \mathbf{0}$ the all zero vector) and adapts it each time a training point is misclassified by the current weights. The algorithm is shown in Table 2.1. The algorithm updates the weight vector and bias directly, something that we will refer to as the primal form in-contrast to an alternative dual representation which we will introduce below.

Table 2.1: The Perceptron Algorithm (primal form)

Given a linearly separable training set S and learning rate $\mathbb{R}^+$

$\mathbf{w}_o \leftarrow \mathbf{0}; b_o \leftarrow 0; k \leftarrow 0$

$R \leftarrow \max_{1 \le i \le \ell} \|\mathbf{x}_i\|$

repeat

for $i = 1$ to ℓ

if $y_i(\langle \mathbf{w}_k \cdot \mathbf{x}_i \rangle + b_k) \le 0$ then

$\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + \eta y_i \mathbf{x}_i$

$b_{k+1} \leftarrow b_k + \eta y_i R^2$

$k \leftarrow k + 1$

end if

end for

until no mistakes made within the *for* loop

return $(\mathbf{w}_k, b_k)$ where k is the number of mistakes

This procedure is guaranteed to converge provided there exists a hyperplane that correctly classifies the training data. In this case we say that the data are *linearly separable.* If no such hyperplane exists the data are said to be nonseparable. Below we will show that the number of iterations depends on a quantity called the margin. This quantity will play a central role throughout the book and so we will introduce a more formal definition here.

Definition 2.2

We define the *(functional) margin of an example* $(\mathbf{x}_i, y_i)$ *with respect to a hyperplane* $(\mathbf{w}, b)$ to be the quantity

$$\gamma_i = y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b).$$

Note that $_i > 0$ implies correct classification of $(\mathbf{x}_i, y_i)$. The *(functional) margin distribution of a hyperplane* $(\mathbf{w}, b)$ *with respect to a training set S* is the distribution of the margins of the examples in *S*. We sometimes refer to the minimum of the margin distribution as the *(functional) margin of a hyperplane* $(\mathbf{w}, b)$ *with respect to a training set S.* In both definitions if we replace functional margin by *geometric margin* we obtain the equivalent quantity for the normalised linear function $\left(\frac{1}{\|\mathbf{w}\|}\mathbf{w}, \frac{1}{\|\mathbf{w}\|}b\right)$, which therefore measures the Euclidean distances of the points from the decision boundary in the input space. Finally, the *margin of a training set S* is the maximum geometric margin over all hyperplanes. A hyperplane realising this maximum is known as a *maximal margin hyperplane.* The size of its margin will be positive for a linearly separable training set.

Figure 2.2 shows the geometric margin at two points with respect to a hyperplane in two dimensions. The geometric margin will equal the functional margin if the weight vector is a unit vector. Figure 2.3 shows the margin of a training set.

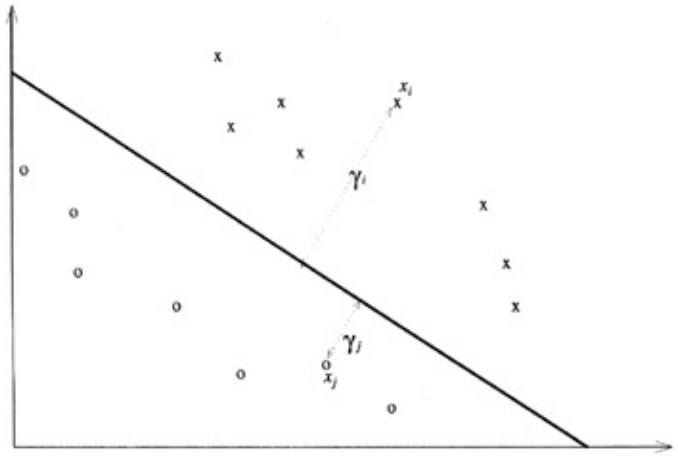


Figure 2.2: The geometric margin of two points

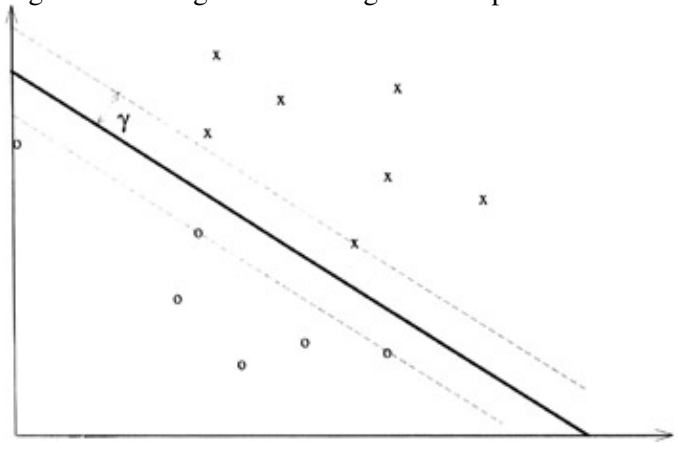


Figure 2.3: The margin of a training set

Theorem 2.3

(Novikoff) Let S be a non-trivial training set, and let

$$R = \max_{1 \le i \le \ell} \|\mathbf{x}_i\| .$$

Suppose that there exists a vector $\mathbf{w}_{opt}$ such that $\|\mathbf{w}_{opt}\| = 1$ and

$$y_i(\langle \mathbf{w}_{opt} \cdot \mathbf{x}_i \rangle + b_{opt}) \ge \gamma$$

for $1 \le i \le$. Then the number of mistakes made by the on-line perceptron algorithm on S is at most

$$\left(\frac{2R}{\gamma}\right)^2 .$$

Proof For the analysis we augment the input vectors by an extra coordinate with value R. We denote the new vector by $\widehat{\mathbf{x}}_i = (\mathbf{x}_i', R)'$, where x denotes the transpose of **x**. Similarly we add an extra coordinate to the weight vector **w** by incorporating the bias b, to form the augmented weight vector $\widehat{\mathbf{w}} = (\mathbf{w}', b/R)'$. The algorithm starts with an augmented weight vector $\widehat{\mathbf{w}}_0 = \mathbf{0}$ and updates it at each mistake. Let $\widehat{\mathbf{w}}_{t-1}$ be the augmented weight vector prior to the tth mistake. The tth update is performed when

$$y_i \langle \widehat{\mathbf{w}}_{t-1} \cdot \widehat{\mathbf{x}}_i \rangle = y_i(\langle \mathbf{w}_{t-1} \cdot \mathbf{x}_i \rangle + b_{t-1}) \le 0$$

where $(\mathbf{x}, y)$ S is the point incorrectly classified by $\widehat{\mathbf{w}}_{t-1} = \left(\mathbf{w}_{t-1}', b_{t-1}/R\right)'$. The update is the following: $_i$

$$\widehat{\mathbf{w}}_t = \left(\mathbf{w}_t', b_t/R\right)' = \left(\mathbf{w}_{t-1}', b_{t-1}/R\right)' + \eta y_i \left(\mathbf{x}_i', R\right)' = \widehat{\mathbf{w}}_{t-1} + \eta y_i \widehat{\mathbf{x}}_i,$$

where we have used the fact that

$$\begin{aligned} b_t/R &= b_{t-1}/R + \eta y_i R \\ \text{since } b_t &= b_{t-1} + \eta y_i R^2. \end{aligned}$$

The derivation

$$\langle \widehat{\mathbf{w}}_t \cdot \widehat{\mathbf{w}}_{\text{opt}} \rangle = \langle \widehat{\mathbf{w}}_{t-1} \cdot \widehat{\mathbf{w}}_{\text{opt}} \rangle + \eta y_i \langle \widehat{\mathbf{x}}_i \cdot \widehat{\mathbf{w}}_{\text{opt}} \rangle \geq \langle \widehat{\mathbf{w}}_{t-1} \cdot \widehat{\mathbf{w}}_{\text{opt}} \rangle + \eta\gamma$$

implies (by induction) that

$$\langle \widehat{\mathbf{w}}_t \cdot \widehat{\mathbf{w}}_{\text{opt}} \rangle \geq t\eta\gamma.$$

Similarly, we have

$$\begin{aligned} \|\widehat{\mathbf{w}}_t\|^2 &= \|\widehat{\mathbf{w}}_{t-1}\|^2 + 2\eta y_i \langle \widehat{\mathbf{w}}_{t-1} \cdot \widehat{\mathbf{x}}_i \rangle + \eta^2 \|\widehat{\mathbf{x}}_i\|^2 \\ &\leq \|\widehat{\mathbf{w}}_{t-1}\|^2 + \eta^2 \|\widehat{\mathbf{x}}_i\|^2 \\ &\leq \|\widehat{\mathbf{w}}_{t-1}\|^2 + \eta^2 \left(\|\mathbf{x}_i\|^2 + R^2 \right) \\ &\leq \|\widehat{\mathbf{w}}_{t-1}\|^2 + 2\eta^2 R^2, \end{aligned}$$

which implies that

$$\|\widehat{\mathbf{w}}_t\|^2 \leq 2t\eta^2 R^2.$$

The two inequalities combined give the ‘squeezing' relations

$$\|\widehat{\mathbf{w}}_{\text{opt}}\| \sqrt{2t}\eta R \geq \|\widehat{\mathbf{w}}_{\text{opt}}\| \, \|\widehat{\mathbf{w}}_t\| \geq \langle \widehat{\mathbf{w}}_t, \widehat{\mathbf{w}}_{\text{opt}} \rangle \geq t\eta\gamma,$$

which together imply the bound

$$t \leq 2 \left(\frac{R}{\gamma} \right)^2 \|\widehat{\mathbf{w}}_{\text{opt}}\|^2 \leq \left(\frac{2R}{\gamma} \right)^2,$$

since $b_{\text{opt}} \leq R$ for a non-trivial separation of the data, and hence

$$\|\widehat{\mathbf{w}}_{\text{opt}}\|^2 \leq \|\mathbf{w}_{\text{opt}}\|^2 + 1 = 2.$$

Remark 2.4

The theorem is usually given for zero bias and the bound is a factor 4 better in this case. However, the bias is updated in the perceptron algorithm and if the standard update is made (without the R^2 factor) the number of iterations depends on the margin of the augmented (including bias) weight training set. This margin is always less than or equal to and can be very much smaller. It will equal when $b_{\text{opt}} = 0$, and so the bound using the augmented margin will be a factor 4 better, though in this case the factor of 2 introduced in the last line of our proof can be avoided, making our bound only a factor 2 worse. In contrast, for cases where $|b_{\text{opt}}| = O(R)$, with $R > 1$, the bound obtained with the augmented training set will be a factor $O(R^2)$ worse than our bound.

Remark 2.5

The critical quantity in the bound is the square of the ratio of the radius of the ball containing the data and the margin of the separating hyperplane. This ratio is invariant under a positive rescaling of the data, and it is clear that rescaling will not affect the number of iterations taken by the algorithm, though it is at first counter-intuitive that this number does not depend on the learning rate. The reason is that there is a degree of freedom in the description of a thresholded linear function, as a positive rescaling of both weights and bias does not change its classification. Later we will use this fact to define the canonical maximal margin hyperplane with respect to a separable training set by fixing the margin equal to 1 and minimising the norm of the weight vector. The resulting value of the norm is inversely proportional to the margin.

The theorem proves that the algorithm converges in a finite number of iterations provided its margin is positive. Just iterating several times on the same sequence S, after a number of mistakes bounded by $\left(\frac{2R}{\gamma}\right)^2$ the perceptron algorithm will find a separating hyperplane and hence halt, provided one exists.

In cases where the data are not linearly separable, the algorithm will not converge: if executed iteratively on the sequence S it will continue to oscillate, changing hypothesis $\mathbf{w}_i$ every time it finds a misclassified point. However, a theorem similar to Novikoff's exists, bounding the number of errors made during one iteration. It uses a different measure of the margin distribution, a measure that will play an important role in later chapters. Intuitively, it generalises the notion of margin to account for a more global property of the training sample, using the margins achieved by training points other than those closest to the hyperplane. This measure of the margin distribution can also be used to measure the amount of 'non-separability' of the sample.

Definition 2.6

Fix a value $\gamma > 0$, we can define the margin slack variable of an example $(\mathbf{x}_i, y_i)$ with respect to the hyperplane $(\mathbf{w},b)$ and target margin γ as

$$\xi\left((\mathbf{x}_i, y_i), (\mathbf{w}, b), \gamma\right) = \xi_i = \max\left(0, \gamma - y_i\left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b\right)\right).$$

Informally this quantity measures how much a point fails to have a margin of γ from the hyperplane. If $\xi_i > \gamma$, then $\mathbf{x}_i$ is misclassified by $(\mathbf{w}, b)$. The norm $\| \ \|_2$ measures the amount by which the training set fails to have margin γ, and takes into account any misclassifications of the training data.

Figure 2.4 shows the size of the margin slack variables for two misclassified points for a hyperplane with unit norm. All of the other points in the figure have their slack variable equal to zero since they have a (positive) margin of more than γ.

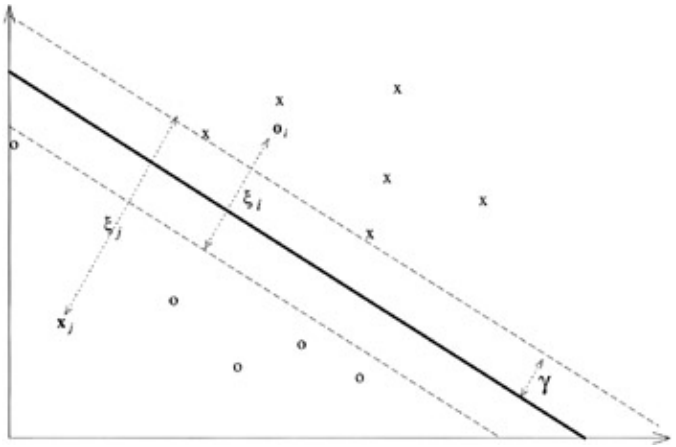


Figure 2.4: The slack variables for a classification problem

Theorem 2.7

(Freund and Schapire) Let S be a non-trivial training set with no duplicate examples, with $\|\mathbf{x}_i\| \leq R$. Let $(\mathbf{w}, b)$ be any hyperplane with $\|\mathbf{w}\| = 1$, let $\gamma > 0$ and define

$$D = \sqrt{\sum_{i=1}^{\ell} \xi_i^2} = \sqrt{\sum_{i=1}^{\ell} \xi\left((\mathbf{x}_i, y_i), (\mathbf{w}, b), \gamma\right)^2}.$$

Then the number of mistakes in the first execution of the for loop of the perceptron algorithm of Table 2.1 on S is bounded by

$$\left(\frac{2(R+D)}{\gamma}\right)^2.$$

Proof The proof defines an extended input space parametrised by Δ in which there is a hyperplane with margin that has the same functionality as $(\mathbf{w}, b)$ on unseen data. We can then apply Theorem 2.3 in the extended space. Finally, optimising the choice of Δ will give the result. The extended input space has an extra coordinate for each training example. The new entries for example $\mathbf{x}_i$ are all zero except for the value Δ in the ith additional coordinate. Let $\tilde{\mathbf{x}}_i$ denote this extended vector and $\tilde{S}$ the corresponding training set. We now extend $\mathbf{w}$ with the value ξ_i/Δ in the ith additional entry to give the vector $\tilde{\mathbf{w}}$. Observe that

$$y_i\left(\langle\tilde{\mathbf{w}}\cdot\tilde{\mathbf{x}}_i\rangle+b\right)=y_i\left(\langle\mathbf{w}\cdot\mathbf{x}_i\rangle+b\right)+\xi_i\geq\gamma,$$

showing that $(\tilde{\mathbf{w}}, b)$ has margin on $\tilde{S}$. Note, however, that $\|\tilde{\mathbf{w}}\|=\sqrt{1+D^2/\Delta^2}$, so that the geometric margin $\tilde{\gamma}$ is reduced by this factor. Since the extended training examples have non-zero entries in different coordinates, running the perceptron algorithm for the first *for* loop on $\tilde{S}$ has the same effect as running it on S, and we can bound the number of mistakes by Theorem 2.3 by

$$\left(\frac{2\tilde{R}}{\tilde{\gamma}}\right)^2=\frac{4\left(R^2+\Delta^2\right)\left(1+D^2/\Delta^2\right)}{\gamma^2}.$$

The bound is optimised by choosing $\Delta=\sqrt{RD}$ giving the required result.

Remark 2.8

The reason we can only apply the theorem for the first iteration of the *for* loop is that after a training example $\tilde{\mathbf{x}}_i$ has been used to update the weight vector in the extended space, the ith additional coordinate of the weight vector will have a non-zero entry, which will affect the evaluation of $\tilde{\mathbf{x}}_i$ when it is processed in subsequent iterations. One could envisage an adaptation of the perceptron algorithm that would include these additional coordinates, though the value of Δ would have to either be a parameter or be estimated as part of the computation.

Remark 2.9

Since D can be defined for any hyperplane, the bound of the theorem does not rely on the data being linearly separable. The problem of finding the linear separation of non-separable data with the smallest number of misclassifications is NP-complete. A number of heuristics have been proposed for this problem, for example the pocket algorithm outputs the $\mathbf{w}$ that survived for the longest number of iterations. The extension suggested in the previous remark could be used to derive a version of the perceptron algorithm for non-separable data.

It is important to note that the perceptron algorithm works by adding misclassified positive training examples or subtracting misclassified negative ones to an initial arbitrary weight vector. Without loss of generality, we have assumed that the initial weight vector is the zero vector, and so the final hypothesis will be a linear combination of the training points:

$$\mathbf{w}=\sum_{i=1}^{\ell}\alpha_i y_i\mathbf{x}_i,$$

where, since the sign of the coefficient of $\mathbf{x}_i$ is given by the classification y_i, the α_i are positive values proportional to the number of times misclassification of $\mathbf{x}_i$ has caused the weight to be updated. Points that have caused fewer mistakes will have smaller α_i, whereas difficult points will have large values. This quantity is sometimes referred to as the *embedding strength* of the pattern $\mathbf{x}_i$, and will play an important role in later

chapters. Once a sample S has been fixed, one can think of the vector as alternative representation of the hypothesis in different or dual coordinates. This expansion is however not unique: different can correspond to the same hypothesis $\mathbf{w}$. Intuitively, one can also regard $_i$ as an indication of the information content of the example $\mathbf{x}_i$. In the case of non-separable data, the coefficients of misclassified points grow indefinitely.

The decision function can be rewritten in dual coordinates as follows:

$$\begin{aligned} h(\mathbf{x}) &= \operatorname{sgn}\left(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b\right) \\ &= \operatorname{sgn}\left(\left\langle \sum_{j=1}^{\ell} \alpha_j y_j \mathbf{x}_j \cdot \mathbf{x} \right\rangle + b\right) \\ &= \operatorname{sgn}\left(\sum_{j=1}^{\ell} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x} \rangle + b\right), \end{aligned} \tag{2.1}$$

and the perceptron algorithm can also be expressed entirely in this dual form as shown in Table 2.2. Note that the learning rate only changes the scaling of the hyperplanes, it does not affect the algorithm with a zero starting vector and so we have no longer included it.

Table 2.2: The Perceptron Algorithm (dual form)

Given training set S

$\mathbf{0}$; b 0

R $\max_{1 \le i \le}$ $\|\mathbf{x}i\|$

repeat

for $i = 1$ to ℓ

if $y_i \left(\sum_{j=1}^{\ell} \alpha_j y_j \langle \mathbf{x}_j \cdot \mathbf{x}_i \rangle + b\right) \le 0$ then

$\alpha_i \leftarrow \alpha_i + 1$

$b \leftarrow b + y_i R^2$

end if

end for

until no mistakes made within the *for* loop

return (,b) to define function $h(\mathbf{x})$ of equation (2.1)

This alternative formulation of the perceptron algorithm and its decision function has many interesting properties. For example, the fact that the points that are harder to learn have larger $_i$ can be used to rank the data according to their information content. Indeed, in the analysis of the simple perceptron algorithm we have already found many of the important concepts that will be used in the theory of Support Vector Machines: the margin, the margin distribution, and the dual representation.

Remark 2.10

Since the number of updates equals the number of mistakes and each update causes 1 to be added to exactly one of its components, the 1-norm of the vector satisfies

$$\|\boldsymbol{\alpha}\|_1 \le \left(\frac{2R}{\gamma}\right)^2,$$

the bound on the number of mistakes given in Theorem 2.3. We can therefore view the 1-norm of as a measure of the complexity of the target concept in the dual representation.

Remark 2.11

The training data only enter the algorithm through the entries of the matrix $\mathbf{G} = (\mathbf{x}_i \mathbf{x}_j)_{i,j=1}$, known as the Gram matrix, whose properties are briefly discussed in Appendix B and which will be related to other similar matrices in later chapters. This observation will have important consequences in Chapter 3.

2.1.2 Other Linear Classifiers

The problem of learning a hyperplane that separates two (separable) sets of points is an ill-posed one, in the sense that in general several different solutions exist. For example the perceptron algorithm may give a different solution depending on the order in which the examples are processed. The danger with ill-posed problems is that not all solutions may be equally useful. One way to render it well-posed is to try to optimise a different cost function, which ensures that if a solution exists it is always unique. For example we can choose not simply to learn any rule that correctly separates the two classes, but to choose from among these rules the one that is a maximum distance from the data. This is the hyperplane that realises the maximal margin. It is also said to have *maximal stability*. An iterative algorithm similar to the perceptron algorithm exists that is guaranteed to converge to the maximal margin solution. We will briefly analyse this algorithm in Chapter 7.

The perceptron algorithm is guaranteed to converge only if the data are linearly separable. A procedure that does not suffer from this limitation is Fisher's discriminant, which is aimed at finding the hyperplane ($\mathbf{w}$, b) on which the projection of the data is maximally separated. The cost function to be optimised is the following:

$$F = \frac{(m_1 - m_{-1})^2}{\sigma_1^2 + \sigma_{-1}^2}$$

where m_i and $_i$ are respectively the mean and standard deviation of the function output values

$$\{\langle \mathbf{w} \cdot \mathbf{x}_j \rangle + b : y_j = i\}$$

for the two classes, $i = 1, -1$.

The hyperplane that optimises this criterion can be found by solving a system of linear equations with a symmetric matrix formed from the training data and right hand side the difference between the two class means.

2.1.3 Multi-class Discrimination

The problem of two-class discrimination we have studied so far can also be solved by defining a weight vector $\mathbf{w}_i$ and a bias b_i for each class. Each time a new instance has to be classified, both functions are evaluated, and the point $\mathbf{x}$ is assigned to class 1 if $\mathbf{w_1} \cdot \mathbf{x} + b_1 \geq \mathbf{w}_{-1} \cdot \mathbf{x} + b_{-1}$, to class -1 otherwise. This approach is equivalent to discrimination using the single hyperplane ($\mathbf{w}$, b), with the substitutions $\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_{-1}$, $b = b_1 - b_{-1}$.

For a multi-class classification problem the output domain is $Y = \{1, 2, \ldots, m\}$. The generalisation of linear learning machines to the m-class case is straightforward: to each of the m classes are associated a weight vector and a bias, ($\mathbf{w}_i$, b_i), i $\{1, \ldots, m\}$, and the decision function is given by

$$c(\mathbf{x}) = \arg \max_{1 \leq i \leq m} \left(\langle \mathbf{w}_i \cdot \mathbf{x} \rangle + b_i \right).$$

Geometrically this is equivalent to associating a hyperplane to each class, and to assigning a new point $\mathbf{x}$ to the class whose hyperplane is furthest from it. The input space is split into m simply connected and convex

regions.

Algorithms for learning the m hyperplanes simultaneously from data exist, and are extensions of the basic procedures outlined above.

2.2 Linear Regression

The problem of linear regression consists in finding a linear function

$$f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b$$

that best interpolates a given set S of training points labelled from $Y \subseteq \mathbb{R}$. Geometrically this corresponds to a hyperplane fitting the given points. Figure 2.5 shows a one dimensional linear regression function. The distance shown as ξ in the figure is the error for the particular training example.

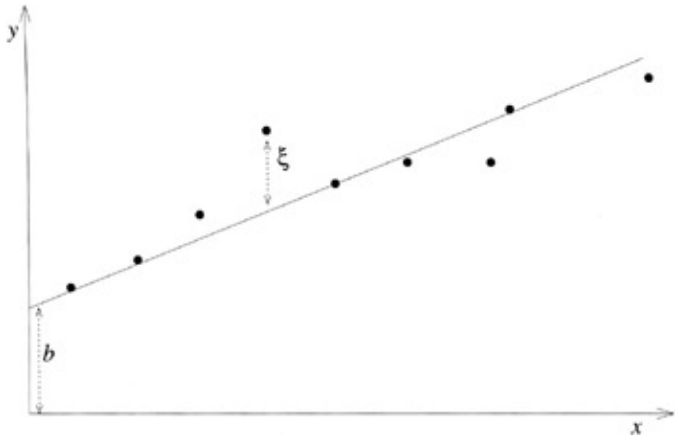


Figure 2.5: A one dimensional linear regression function

This problem has been studied since the 18th century, and the best-known solution is that proposed independently by Gauss and Legendre of choosing the line that minimises the sum of the squares of the distances from the training points. This technique is known as least squares, and is known to be optimal in the case of linear targets corrupted by Gaussian noise.

Numerical stability and generalisation considerations motivated the introduction of a variation of this technique, which is analogous to the maximal margin hyperplane in the classification case: choosing a function that minimises a combination of square loss and norm of the $\mathbf{w}$ vector. This solution, proposed by Hoerl and Kennard, is known as *ridge regression.* Both these algorithms require the inversion of a matrix, though a simple iterative procedure also exists (the Adaline algorithm developed by Widrow and Hoff in the 1960s). Note that these regression techniques can also be used for classification problems, with a careful choice of the target values associated with the classes.

2.2.1 Least Squares

Given a training set S, with $\mathbf{x}_i \in X \subseteq \mathbb{R}^n$, $y_i \in Y \subseteq \mathbb{R}$, the problem of linear regression is to find a (linear) function f that models the data

$$y = f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x} \rangle + b.$$

The least squares approach prescribes choosing the parameters $(\mathbf{w}, b)$ to minimise the sum of the squared deviations of the data,

$$L(\mathbf{w}, b) = \sum_{i=1}^{\ell} \left(y_i - \langle \mathbf{w} \cdot \mathbf{x}_i \rangle - b \right)^2.$$

The function L is known as the square loss function as it measures the amount of loss associated with the particular choice of parameters by a sum of squares. The loss can also be measured using other loss functions. The use of the symbol L should not be confused with its use for the Lagrangian introduced in Chapter 5. We can minimise L by differentiating with respect to the parameters $(\mathbf{w},b)$, and setting the resulting $n + 1$ linear

expressions to zero. This is best expressed in matrix notation by setting $\widehat{\mathbf{w}} = (\mathbf{w}', b)'$, and

$$\widehat{\mathbf{X}} = \begin{pmatrix} \widehat{\mathbf{x}}_1' \\ \widehat{\mathbf{x}}_2' \\ \vdots \\ \widehat{\mathbf{x}}_\ell' \end{pmatrix}, \text{ where } \widehat{\mathbf{x}}_i = (\mathbf{x}_i', 1)'.$$

With this notation the vector of output discrepancies becomes

$$\mathbf{y} - \widehat{\mathbf{X}}\widehat{\mathbf{w}}$$

with **y** a column vector. Hence, the loss function can be written as

$$L(\widehat{\mathbf{w}}) = (\mathbf{y} - \widehat{\mathbf{X}}\widehat{\mathbf{w}})'(\mathbf{y} - \widehat{\mathbf{X}}\widehat{\mathbf{w}}).$$

Taking derivatives of the loss and setting them equal to zero,

$$\frac{\partial L}{\partial \widehat{\mathbf{w}}} = -2\widehat{\mathbf{X}}'\mathbf{y} + 2\widehat{\mathbf{X}}'\widehat{\mathbf{X}}\widehat{\mathbf{w}} = \mathbf{0},$$

yields the well-known 'normal equations'

$$\widehat{\mathbf{X}}'\widehat{\mathbf{X}}\widehat{\mathbf{w}} = \widehat{\mathbf{X}}'\mathbf{y},$$

and, if the inverse of $\widehat{\mathbf{X}}'\widehat{\mathbf{X}}$ exists, the solution of the least squares problem is

$$\widehat{\mathbf{w}} = (\widehat{\mathbf{X}}'\widehat{\mathbf{X}})^{-1}\widehat{\mathbf{X}}'\mathbf{y}.$$

If $\widehat{\mathbf{X}}'\widehat{\mathbf{X}}$ is singular, the pseudo-inverse can be used, or else the technique of ridge regression described below can be applied.

In the 1960s attention was paid to the construction of simple iterative procedures for training linear learning machines. The Widrow-Hoff algorithm (also known as Adaline) converges to this least squares solution and has a similar flavour to the perceptron algorithm but implements a simple gradient descent strategy. The algorithm is shown in Table 2.3.

Table 2.3: The Widrow-Hoff Algorithm (primal form)

Given training set S and learning rate $\mathbb{R}^+$

$\mathbf{w}_0 \quad \mathbf{0}$; $b \quad 0$

repeat

for $i = 1$ to ℓ

$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta\left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b - y_i\right)(\mathbf{x}_i, 1)$

end for

until convergence criterion satisfied

return $(\mathbf{w}, b)$

2.2.2 Ridge Regression

If the matrix $\widehat{\mathbf{X}}'\widehat{\mathbf{X}}$ in the least squares problem is not of full rank, or in other situations where numerical stability problems occur, one can use the following solution:

$$\widehat{\mathbf{w}} = (\widehat{\mathbf{X}}'\widehat{\mathbf{X}} + \lambda \mathbf{I}_n)^{-1}\widehat{\mathbf{X}}'\mathbf{y}$$

obtained by adding a multiple $\mathbb{R}^+$ of the diagonal matrix $\mathbf{I}_n$ to the matrix $\widehat{\mathbf{X}}'\widehat{\mathbf{X}}$, where $\mathbf{I}_n$ is the identity matrix with the $(n+1, n+1)$ entry set to zero. This solution is called *ridge regression,* and was originally motivated by statistical as well as numerical considerations.

The ridge regression algorithm minimises the *penalised loss function*

$$L(\mathbf{w}, b) = \lambda \langle \mathbf{w} \cdot \mathbf{w} \rangle + \sum_{i=1}^{\ell} \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b - y_i \right)^2 \qquad (2.2)$$

so that the parameter controls a trade-off between low square loss and low norm of the solution.

Remark 2.12

This algorithm is analogous to the maximal margin algorithm in the classification case, presenting a complex cost function that is made of two parts, one controlling the 'complexity' of the hypothesis, and the other its accuracy on the training data. In Chapter 4 we will present a systematic study of this kind of cost function, and of the generalisation properties of linear learning machines motivated by it.

Note that ridge regression also admits a dual representation. The solution needs to satisfy $\frac{\partial L}{\partial \mathbf{w}} = \mathbf{0}$, which gives the following expression for the hypothesis: $\mathbf{w} = -\sum_i (\mathbf{w} \cdot \mathbf{x}_i \quad b - y_i)\mathbf{x}_i$, which implies that there exist scalars $\alpha_i = -\frac{1}{\lambda}\left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle b - y_i \right)$, such that the solution can be written as $\mathbf{w} = \sum_{i \ i \ i} \mathbf{x}$.

Once we know that the solution can be expressed in dual form we can derive conditions that must satisfy. We can express the duality condition in vector form by expressing the weight vector in terms of the vector :

$$\mathbf{w} = \mathbf{X}'\boldsymbol{\alpha},$$

where $\mathbf{X}$ is the matrix $\widehat{\mathbf{X}}$ with the last column (of 1s) removed. We can rewrite equation (2.2) as follows, where we have set b to zero for simplicity:

$$\begin{aligned} L(\mathbf{w}) &= \lambda \boldsymbol{\alpha}'\mathbf{X}\mathbf{X}'\boldsymbol{\alpha} + \sum_{i=1}^{\ell} \left(\boldsymbol{\alpha}'\mathbf{X}\mathbf{x}_i - y_i \right)^2 \\ &= \lambda \boldsymbol{\alpha}'\mathbf{G}\boldsymbol{\alpha} + \sum_{i=1}^{\ell} \left((\mathbf{G}\boldsymbol{\alpha})_i - y_i \right)^2 \\ &= \lambda \boldsymbol{\alpha}'\mathbf{G}\boldsymbol{\alpha} + (\mathbf{G}\boldsymbol{\alpha} - \mathbf{y})'(\mathbf{G}\boldsymbol{\alpha} - \mathbf{y}) \\ &= \lambda \boldsymbol{\alpha}'\mathbf{G}\boldsymbol{\alpha} + \boldsymbol{\alpha}'\mathbf{G}\mathbf{G}\boldsymbol{\alpha} - 2\mathbf{y}'\mathbf{G}\boldsymbol{\alpha} + \mathbf{y}'\mathbf{y} \end{aligned}$$

where $\mathbf{G} = \mathbf{X}\mathbf{X} \quad = \mathbf{G}$. Taking derivatives with respect to and setting to zero we obtain the equation

$$2\mathbf{G}(\lambda \boldsymbol{\alpha} + \mathbf{G}\boldsymbol{\alpha} - \mathbf{y}) = \mathbf{0}$$

This equation will be satisfied if

$$(\lambda \mathbf{I} + \mathbf{G})\boldsymbol{\alpha} = \mathbf{y}$$

giving a predictive function of

$$f(\mathbf{x}) = \mathbf{y}'(\lambda \mathbf{I} + \mathbf{G})^{-1}\mathbf{z}$$

where $z_i = \quad \mathbf{x} \cdot \mathbf{x}_i$. Note how this dual equation depends on the Gram matrix of inner products of the training examples, $\mathbf{G} = \mathbf{X}\mathbf{X}$.

---

4 Chapter 2: Linear Learning Machines

2.3 Dual Representation of Linear Machines

In the previous sections we have stressed that for most of the linear machines described there exists a dual description. This representation will be used in subsequent chapters, and will be shown to be a general property of a wide class of algorithms. Duality will be one of the crucial concepts in developing Support Vector Machines.

An important property of the dual representation is that the data only appear through entries in the Gram matrix and never through their individual attributes. Similarly in the dual representation of the decision function, *it is only the inner products of the data with the new test point* that are needed. This fact will have far reaching consequences in the rest of the book.

Finally note that in Chapter 5 we will provide a systematic view of many of the issues concerning duality that we have touched on ‘empirically' in this chapter. Many of the problems and algorithms discussed here will be shown to be special cases of optimisation problems, for which a mathematical framework exists that naturally encompasses duality.

2.4 Exercises

1. Write the Widrow-Hoff algorithm in dual form.
2. Write an iterative algorithm for linear regression in primal and dual form.
3. Develop a version of the perceptron algorithm for non-separable data using the approach described in Remarks 2.8 and 2.9.

2.5 Further Reading and Advanced Topics

The theory of linear discriminants dates back to the 1930s, when Fisher [40] proposed a procedure for classification. In the field of artificial intelligence attention was drawn to this problem by the work of Frank Rosenblatt [122], who starting from 1956 introduced the perceptron learning rule. Minsky and Papert's famous book Perceptrons [98] analysed the computational limitations of linear learning machines. The famous book by Duda and Hart [35] provides a complete survey of the state of the art up to 1973. For more recent results, see also [16] which includes a description of a class of generalised learning machines.

The extension of Novikoff's theorem [104] to the non-separable case is due to [43]; the pocket algorithm was proposed by Gallant [47]; a simple description of Fisher's discriminant is in [35]. For discussions of the computational complexity of learning linear separations in the non-separable case, see [64] and [8]. The idea of a maximal margin hyperplane has been rediscovered several times. It is discussed by Vapnik and Lerner in [166], by Duda and Hart [35], and an iterative algorithm known as *Adatron* for learning maximal margin hyperplanes was proposed in the statistical mechanics literature by [4], and will be further discussed in Chapter 7.

The problem of linear regression is much older than the classification one. Least squares linear interpolation was first used by Gauss in the 18th century for astronomical problems. The ridge regression algorithm was published by Hoerl and Kennard [63], and subsequently discovered to be a special case of the regularisation theory of Tikhonov [153] for the solution of ill-posed problems. The dual form of ridge regression including the derivations of Subsection 2.2.2 was studied by Saunders et al. [125] and [144]. An equivalent heuristic was widely used in the neural networks literature under the name of weight decay. The Widrow-Hoff algorithm is described in [179].

Finally note that the representation of linear machines in the dual form, using the training data, is intimately related to the optimisation technique of Lagrange multipliers, and will be further discussed in Chapter 5. Guyon and Stork [56] compare the primal and dual representation of linear learning machines in an analogous way to that adopted in this chapter.

These references are also given on the website http://www.support-vector.net, which will be kept up to date with new work, pointers to software and papers that are available on-line.

Chapter 3: Kernel-Induced Feature Spaces

Overview

The limited computational power of linear learning machines was highlighted in the 1960s by Minsky and Papert. In general, complex real-world applications require more expressive hypothesis spaces than linear functions. Another way of viewing this problem is that frequently the target concept cannot be expressed as a simple linear combination of the given attributes, but in general requires that more abstract features of the data be exploited. Multiple layers of thresholded linear functions were proposed as a solution to this problem, and this approach led to the development of multi-layer neural networks and learning algorithms such as back-propagation for training such systems.

Kernel representations offer an alternative solution by projecting the data into a high dimensional feature space to increase the computational power of the linear learning machines of Chapter 2. The use of linear machines in the dual representation makes it possible to perform this step implicitly. As noted in Chapter 2, the training examples never appear isolated but always in the form of inner products between pairs of examples. The advantage of using the machines in the dual representation derives from the fact that in this representation the number of tunable parameters does not depend on the number of attributes being used. By replacing the inner product with an appropriately chosen ‘kernel' function, one can implicitly perform a non-linear mapping to a high dimensional feature space without increasing the number of tunable parameters, provided the kernel computes the inner product of the feature vectors corresponding to the two inputs.

In this chapter we will introduce the kernel technique, which provides one of the main building blocks of Support Vector Machines. One of the remarkable features of SVMs is that to a certain extent the approximation-theoretic issues are independent of the learning-theoretic ones. One can therefore study the properties of the kernel representations in a general and self-contained way, and use them with different learning theories, as we will see in Chapter 4.

Another attraction of the kernel method is that the learning algorithms and theory can largely be decoupled from the specifics of the application area, which must simply be encoded into the design of an appropriate kernel function. Hence, the problem of choosing an architecture for a neural network application is replaced by the problem of choosing a suitable kernel for a Support Vector Machine. In this chapter we will describe some well-known kernels and show how more complicated kernels can be constructed by combining simpler ones. We will also mention kernels that have been developed for discrete structures such as text, showing that the approach is not restricted only to input spaces that are subsets of Euclidean space, and hence can be applied even when we were unable to define linear functions over the input space.

As we will see in Chapters 4 and 7, the use of kernels can overcome the curse of dimensionality in both computation and generalisation.

2 Chapter 3: Kernel-Induced Feature Spaces

3.1 Learning in Feature Space

The complexity of the target function to be learned depends on the way it is represented, and the difficulty of the learning task can vary accordingly. Ideally a representation that matches the specific learning problem should be chosen. So one common preprocessing strategy in machine learning involves changing the representation of the data:

$$\mathbf{x} = (x_1, \ldots, x_n) \longmapsto \boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_N(\mathbf{x})).$$

This step is equivalent to mapping the input space X into a new space, $F = \{ \quad (\mathbf{x}) | \mathbf{x} \quad X$.

Example 3.1

Consider the target function

$$f(m_1, m_2, r) = C\frac{m_1 m_2}{r^2},$$

giving Newton's law of gravitation, expressing the gravitational force between two bodies with masses m_1,m_2 and separation r. This law is expressed in terms of the observable quantities, mass and distance. A linear machine such as those described in Chapter 2 could not represent it as written, but a simple change of coordinates

$$(m_1, m_2, r) \longmapsto (x, y, z) = (\ln m_1, \ln m_2, \ln r)$$

gives the representation

$$g(x, y, z) = \ln f(m_1, m_2, r) = \ln C + \ln m_1 + \ln m_2 - 2\ln r = c + x + y - 2z,$$

which could be learned by a linear machine.

The fact that simply mapping the data into another space can greatly simplify the task has been known for a long time in machine learning, and has given rise to a number of techniques for selecting the best representation of data. The quantities introduced to describe the data are usually called *features*, while the original quantities are sometimes called *attributes.* The task of choosing the most suitable representation is known as *feature selection.* The space X is referred to as the input space, while $F = \{ \quad (\mathbf{x}) : |\mathbf{x} \quad X\}$ is called the *feature space.*

Figure 3.1 shows an example of a feature mapping from a two dimensional input space to a two dimensional feature space, where the data cannot be separated by a linear function in the input space, but can be in the feature space. The aim of this chapter is to show how such mappings can be made into very high dimensional spaces where linear separation becomes much easier.

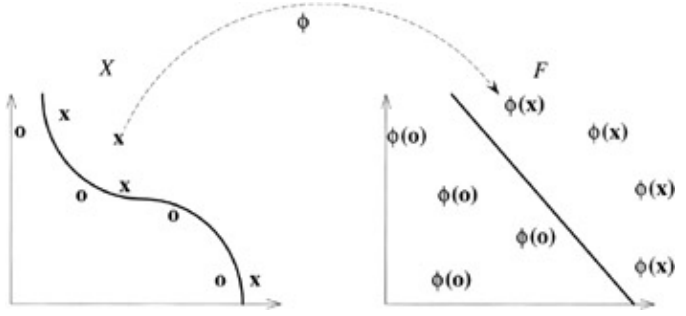


Figure 3.1: A feature map can simplify the classification task

Different approaches to feature selection exist. Frequently one seeks to identify the smallest set of features that still conveys the essential information contained in the original attributes. This is known as *dimensionality reduction*,

$$\mathbf{x} = (x_1, \ldots, x_n) \longmapsto \boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_d(\mathbf{x})), \quad d < n,$$

and can be very beneficial as both computational and generalisation performance can degrade as the number of features grows, a phenomenon sometimes referred to as the *curse of dimensionality*. The difficulties with high dimensional feature spaces are unfortunate, since the larger the set of (possibly redundant) features, the more likely that the function to be learned can be represented using a standardised learning machine. We will show how this degradation of performance can be avoided in the Support Vector Machine.

Example 3.2

Again consider the gravitation law for two bodies, but suppose the attributes are now the three components of their positions together with their masses:

$$\mathbf{x} = (p_1^x, p_1^y, p_1^z, p_2^x, p_2^y, p_2^z, m_1, m_2).$$

One way to reduce the dimensionality of the problem would be the following mapping $\mathbb{R}^8 \quad \mathbb{R}^3$:

$$\mathbf{x} = (p_1^x, p_1^y, p_1^z, p_2^x, p_2^y, p_2^z, m_1, m_2) \mapsto \boldsymbol{\phi}(\mathbf{x}) = \left(\sqrt{\sum_{i \in \{x,y,z\}} (p_1^i - p_2^i)^2}, m_1, m_2 \right),$$

which would retain the essential information.

Another very different feature selection task is the detection of *irrelevant features* and their subsequent elimination. In our example, an irrelevant feature would be the colour of the two bodies, or their temperature, since neither quantity affects the target output value.

The use of principal components analysis provides a mapping of the data to a feature space in which the new features are linear functions of the original attributes and are sorted by the amount of variance that the data exhibit in each direction. Dimensionality reduction can sometimes be performed by simply removing features corresponding to directions in which the data have low variance, though there is no guarantee that these features are not essential for performing the target classification. We now give an example where additional feature dimensions can be useful.

Example 3.3

Consider the case of a two dimensional input space, and assume our prior knowledge about the problem suggests that relevant information is encoded in the form of monomials of degree 2. Hence we want to represent the problem in a feature space where such information is made explicit, and is ready for the learning machine to use. A possible mapping is the following:

$$(x_1, x_2) \longmapsto \boldsymbol{\phi}(x_1, x_2) = (x_1^2, x_2^2, x_1 x_2).$$

In the same way we might want to use features of degree d, giving a feature space of $\binom{n+d-1}{d}$ dimensions, a number that soon becomes computationally infeasible for reasonable numbers of attributes and feature degrees. The use of this type of feature space will require a special technique, introduced in Section 3.2, involving an ‘implicit mapping' into the feature space.

The computational problems are not the only ones connected with the size of the feature space we are using. Another source of difficulties is the generalisation of the learning machine, which can be sensitive to the dimensionality of the representation for standard function classes of hypotheses.

It is evident from the previous examples that feature selection should be viewed as a part of the learning process itself, and should be automated as much as possible. On the other hand, it is a somewhat arbitrary step, which reflects our prior expectations on the underlying target function. The theoretical models of learning should also take account of this step: using too large a set of features can create overfitting problems,

unless the generalisation can be controlled in some way. It is for this reason that research has frequently concentrated on dimensionality reduction techniques. However, we will see in Chapter 4 that a deeper understanding of generalisation means that we can even afford to use infinite dimensional feature spaces. The generalisation problems will be avoided by using learning machines based on this understanding, while computational problems are avoided by means of the ‘implicit mapping' described in the next section.

4 Chapter 3: Kernel-Induced Feature Spaces

3.2 The Implicit Mapping into Feature Space

In order to learn non-linear relations with a linear machine, we need to select a set of non-linear features and to rewrite the data in the new representation. This is equivalent to applying a fixed non-linear mapping of the data to a feature space, in which the linear machine can be used. Hence, the set of hypotheses we consider will be functions of the type

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi_i(\mathbf{x}) + b,$$

where : X F is a non-linear map from the input space to some feature space. This means that we will build non-linear machines in two steps: first a fixed non-linear mapping transforms the data into a feature space F, and then a linear machine is used to classify them in the feature space.

As shown in Chapter 2, one important property of linear learning machines is that they can be expressed in a dual representation. This means that the hypothesis can be expressed as a linear combination of the training points, so that the decision rule can be evaluated using just inner products between the test point and the training points:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i y_i \langle \boldsymbol{\phi}(\mathbf{x}_i) \cdot \boldsymbol{\phi}(\mathbf{x}) \rangle + b.$$

If we have a way of computing the inner product $(\mathbf{x}_i \cdot (\mathbf{x})$ in feature space directly as a function of the original input points, it becomes possible to merge the two steps needed to build a non-linear learning machine. We call such a direct computation method a *kernel* function.

Definition 3.4

A *kernel* is a function K, such that for all **x, z** X

$$K(\mathbf{x}, \mathbf{z}) = \langle \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{z}) \rangle,$$

where is a mapping from X to an (inner product) feature space F.

The name ‘kernel' is derived from integral operator theory, which underpins much of the theory of the relation between kernels and their corresponding feature spaces. An important consequence of the dual representation is that the dimension of the feature space need not affect the computation. As one does not represent the feature vectors explicitly, the number of operations required to compute the inner product by evaluating the kernel function is not necessarily proportional to the number of features. The use of kernels makes it possible to map the data implicitly into a feature space and to train a linear machine in such a space, potentially side-stepping the computational problems inherent in evaluating the feature map. The only information used about the training examples is their Gram matrix (see Remark 2.11) in the feature space. This matrix is also referred to as the *kernel matrix*, and in this context we will use the symbol **K** to denote it. The key to this approach is finding a kernel function that can be evaluated efficiently. Once we have such a function the decision rule can be evaluated by at most evaluations of the kernel:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b.$$

One of the curious facts about using a kernel is that we do not need to know the underlying feature map in order to be able to learn in the feature space! The rest of this chapter will be concerned with the problem of creating such kernel functions. We will consider the properties that they must satisfy as well as some of the

more recent methods developed for their construction. The concept of a kernel is central to the development of the book, but it is not an immediately intuitive idea. First note that the idea of a kernel generalises the standard inner product in the input space. It is clear that this inner product provides an example of a kernel by making the feature map the identity

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle.$$

We can also take the feature map to be any fixed linear transformation $\mathbf{x} \quad \mathbf{Ax}$, for some matrix $\mathbf{A}$. In this case the kernel function is given by

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{Ax} \cdot \mathbf{Az} \rangle = \mathbf{x}'\mathbf{A}'\mathbf{Az} = \mathbf{x}'\mathbf{Bz},$$

where by construction $\mathbf{B} = \mathbf{A} \quad \mathbf{A}$ is a square symmetric positive semi-definite matrix. As discussed in the introduction the aim is to introduce non-linearity into the feature space map. We therefore move to a simple but illustrative example of such a non-linear map obtained by considering the following relation:

$$\begin{aligned} \langle \mathbf{x} \cdot \mathbf{z} \rangle^2 &= \left(\sum_{i=1}^{n} x_i z_i \right)^2 = \left(\sum_{i=1}^{n} x_i z_i \right) \left(\sum_{j=1}^{n} x_j z_j \right) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j z_i z_j = \sum_{(i,j)=(1,1)}^{(n,n)} \left(x_i x_j \right) \left(z_i z_j \right), \end{aligned}$$

which is equivalent to an inner product between the feature vectors

$$\boldsymbol{\phi}(\mathbf{x}) = \left(x_i x_j \right)_{(i,j)=(1,1)}^{(n,n)}.$$

In this case the features are all the monomials of degree 2 considered in Example 3.3, though note that when $i \neq j$ the feature $x_i x_j$ occurs twice, giving it double the weight of the features x^2_i. A more general feature space is obtained by considering the kernel

$$\begin{aligned} \left(\langle \mathbf{x} \cdot \mathbf{z} \rangle + c \right)^2 &= \left(\sum_{i=1}^{n} x_i z_i + c \right) \left(\sum_{j=1}^{n} x_j z_j + c \right) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j z_i z_j + 2c \sum_{i=1}^{n} x_i z_i + c^2 \\ &= \sum_{(i,j)=(1,1)}^{(n,n)} \left(x_i x_j \right) \left(z_i z_j \right) + \sum_{i=1}^{n} \left(\sqrt{2c} x_i \right) \left(\sqrt{2c} z_i \right) + c^2, \end{aligned}$$

whose $\binom{n+1}{2} + n + 1 = \binom{n+2}{2}$ features are all the monomials of degree up to 2, but with the relative weightings between the degree 1 and 2 features controlled by the parameter c, which also determines the strength of the degree 0 or constant feature. Similar derivations can be made for the kernel functions

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle^d \text{ and } K(\mathbf{x}, \mathbf{z}) = \left(\langle \mathbf{x} \cdot \mathbf{z} \rangle + c \right)^d,$$

for $d \geq 2$. For the first kernel the $\binom{n+d-1}{d}$ distinct features are all the monomials of degree d, though again the weighting will vary according to the structure of the exponents. For the second kernel there are $\binom{n+d}{d}$ distinct features, being all the monomials up to and including degree d. The decision boundary in the input space corresponding to a hyperplane in these feature spaces is a polynomial curve of degree d, so these kernels are frequently called *polynomial kernels.*

More complex kernels are possible, and the next section is concerned with the important problem of the construction of kernels. A significant aspect of this problem is the mathematical characterisation of functions $K(\mathbf{x},\mathbf{z})$ that constitute kernels. We will see that a theorem from functional analysis provides an answer to this question.

4 Chapter 3: Kernel-Induced Feature Spaces

3.3 Making Kernels

The use of a kernel function is an attractive computational short-cut. If we wish to use this approach, there appears to be a need to first create a complicated feature space, then work out what the inner product in that space would be, and finally find a direct method of computing that value in terms of the original inputs. In practice the approach taken is to define a kernel function directly, hence implicitly defining the feature space. In this way, we avoid the feature space not only in the computation of inner products, but also in the design of the learning machine itself. We will argue that defining a kernel function for an input space is frequently more natural than creating a complicated feature space. Before we can follow this route, however, we must first determine what properties of a function $K(\mathbf{x},\mathbf{z})$ are necessary to ensure that it is a kernel for some feature space. Clearly, the function must be symmetric,

$$K(\mathbf{x},\mathbf{z}) = \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle = \langle \phi(\mathbf{z}) \cdot \phi(\mathbf{x}) \rangle = K(\mathbf{z},\mathbf{x}),$$

and satisfy the inequalities that follow from the Cauchy-Schwarz inequality,

$$\begin{aligned} K(\mathbf{x},\mathbf{z})^2 &= \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{z}) \rangle^2 \leq \|\phi(\mathbf{x})\|^2 \|\phi(\mathbf{z})\|^2 \\ &= \langle \phi(\mathbf{x}) \cdot \phi(\mathbf{x}) \rangle \langle \phi(\mathbf{z}) \cdot \phi(\mathbf{z}) \rangle = K(\mathbf{x},\mathbf{x}) K(\mathbf{z},\mathbf{z}). \end{aligned}$$

These conditions are, however, not sufficient to guarantee the existence of a feature space.

3.3.1 Characterisation of Kernels

Mercer's Theorem

In this subsubsection we will introduce Mercer's theorem, which provides a characterisation of when a function $K(\mathbf{x},\mathbf{z})$ is a kernel. We begin by considering a simple case in order to motivate the result. Consider a finite input space $X = \{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$, and suppose $K(\mathbf{x},\mathbf{z})$ is a symmetric function on X. Consider the matrix

$$\mathbf{K} = \left(K(\mathbf{x}_i,\mathbf{x}_j)\right)_{i,j=1}^n.$$

Since $\mathbf{K}$ is symmetric there is an orthogonal matrix $\mathbf{V}$ such that $\mathbf{K} = \mathbf{V}\ \ \mathbf{V}'$, where $\quad$ is a diagonal matrix containing the eigenvalues $\ _t$ of $\mathbf{K}$, with corresponding eigenvectors $\mathbf{v}_t = (v_{ti})_{i=1}^n$ the columns of $\mathbf{V}$. Now assume that all the eigenvalues are non-negative and consider the feature mapping

$$\phi : \mathbf{x}_i \mapsto \left(\sqrt{\lambda_t} v_{ti}\right)_{t=1}^n \in \mathbb{R}^n,\ i = 1,\ldots,n.$$

We now have that

$$\langle \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \rangle = \sum_{t=1}^n \lambda_t v_{ti} v_{tj} = \left(\mathbf{V}\Lambda\mathbf{V}'\right)_{ij} = \mathbf{K}_{ij} = K(\mathbf{x}_i,\mathbf{x}_j),$$

implying that $K(\mathbf{x},\mathbf{z})$ is indeed a kernel function corresponding to the feature mapping $\ $. The requirement that the eigenvalues of $\mathbf{K}$ be non-negative is necessary since if we have a negative eigenvalue $\ _s$ with eigenvector $\mathbf{v}_s$, the point

$$\mathbf{z} = \sum_{i=1}^n v_{si}\phi(\mathbf{x}_i) = \sqrt{\Lambda}\mathbf{V}'\mathbf{v}_s$$

in the feature space would have norm squared

$$\|\mathbf{z}\|^2 = \langle \mathbf{z} \cdot \mathbf{z} \rangle = \mathbf{v}_s'\mathbf{V}\sqrt{\Lambda}\sqrt{\Lambda}\mathbf{V}'\mathbf{v}_s = \mathbf{v}_s'\mathbf{V}\Lambda\mathbf{V}'\mathbf{v}_s = \mathbf{v}_s'\mathbf{K}\mathbf{v}_s = \lambda_s < 0,$$

contradicting the geometry of that space. Hence, we have proved by contradiction the following proposition.

Proposition 3.5

Let X be a finite input space with $K(\mathbf{x},\mathbf{z})$ a symmetric function on X. Then $K(\mathbf{x},\mathbf{z})$ is a kernel function if and only if the matrix

$$\mathbf{K} = \left(K(\mathbf{x}_i, \mathbf{x}_j)\right)_{i,j=1}^{n},$$

is positive semi-definite (has non-negative eigenvalues).

Motivated by this simple example, we will allow a slight generalisation of an inner product in a Hilbert space by introducing a weighting λ_i for each dimension,

$$\langle \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{z}) \rangle = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{z}) = K(\mathbf{x}, \mathbf{z}),$$

so that the feature vector becomes

$$\boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots, \phi_n(\mathbf{x}), \ldots).$$

Mercer's theorem gives necessary and sufficient conditions for a continuous symmetric function $K(\mathbf{x},\mathbf{z})$ to admit such a representation

$$K(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{z}),$$

with non-negative λ_i, which is equivalent to $K(\mathbf{x},\mathbf{z})$ being an inner product in the feature space $F \quad (X)$, where F is the l_2 space of all sequences

$$\boldsymbol{\psi} = (\psi_1, \psi_2, \ldots, \psi_i, \ldots)$$

for which

$$\sum_{i=1}^{\infty} \lambda_i \psi_i^2 < \infty.$$

This will implicitly induce a space defined by the feature vector and as a consequence a linear function in F like those described in Chapter 2 will be represented by

$$f(\mathbf{x}) = \sum_{i=1}^{\infty} \lambda_i \psi_i \phi_i(\mathbf{x}) + b = \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}, \mathbf{x}_j) + b,$$

where the first expression is the primal representation of the function, and the second is the dual, the relation between the two being given by

$$\boldsymbol{\psi} = \sum_{j=1}^{\ell} \alpha_j y_j \boldsymbol{\phi}(\mathbf{x}_j).$$

Note that in the primal representation the number of terms in the summation is equal to the dimensionality of the feature space, while in the dual there are $\quad$ terms (the sample size). According to the size of the feature space being considered, one or other of the representations can be more convenient. It is clear from its form that this function is non-linear whenever the kernel function is non-linear.

The analogy with the finite case is very close. The contribution from functional analysis comes from studying the eigenvalue problem for integral equations of the form

$$\int_X K(\mathbf{x},\mathbf{z})\phi(\mathbf{z})d\mathbf{z} = \lambda\phi(\mathbf{x}),$$

where $K(\mathbf{x},\mathbf{z})$ is a bounded, symmetric, and positive kernel function and X is a compact space. We will not go into the details of the analysis but simply quote the theorem (see Appendix B.3 for the notation and definitions).

Theorem 3.6

(Mercer) Let X be a compact subset of $\mathbb{R}^n$. Suppose K is a continuous symmetric function such that the integral operator $T_K : L_2(X) \to L_2(X)$,

$$(T_K f)(\cdot) = \int_X K(\cdot,\mathbf{x})f(\mathbf{x})d\mathbf{x},$$

is positive, that is

$$\int_{X\times X} K(\mathbf{x},\mathbf{z})f(\mathbf{x})f(\mathbf{z})d\mathbf{x}d\mathbf{z} \geq 0,$$

for all $f \in L_2(X)$. Then we can expand $K(\mathbf{x},\mathbf{z})$ in a uniformly convergent series (on $X \times X$) in terms of T_K's eigen-functions $\phi_j \in L_2(X)$, normalised in such a way that $\|\phi_j\|_{L_2} = 1$, and positive associated eigenvalues $\lambda_j \geq 0$,

$$K(\mathbf{x},\mathbf{z}) = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{x})\phi_j(\mathbf{z}).$$

Remark 3.7

The positivity condition

$$\int_{X\times X} K(\mathbf{x},\mathbf{z})f(\mathbf{x})f(\mathbf{z})d\mathbf{x}d\mathbf{z} \geq 0, \forall f \in L_2(X),$$

corresponds to the positive semi-definite condition in the finite case. The finite matrix condition on a set of points $\{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$ is recovered by choosing f to be the weighted sums of delta functions at each $\mathbf{x}_i$. Since such functions are limits of functions in $L_2(X)$, the above condition implies that for any finite subset of X the corresponding matrix is positive semi-definite. The converse is also true since if the positivity condition does not hold for some function f, we can approximate the integral with a finite sum over a mesh of inputs, which if chosen sufficiently finely will still give a negative value. The values of f on the chosen mesh $\{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$ form a vector $\mathbf{v}$, for which the corresponding kernel matrix $\mathbf{K}$ satisfies

$$\mathbf{v}'\mathbf{K}\mathbf{v} < 0,$$

showing that $\mathbf{K}$ fails to be positive semi-definite. The conditions for Mercer's theorem are therefore equivalent to requiring that for any finite subset of X, the corresponding matrix is positive semi-definite. This gives a second characterisation of a kernel function, one that will prove most useful when we come to constructing kernels. We use the term kernel to refer to functions satisfying this property, but in the literature these are often called Mercer kernels.

Remark 3.8

Following the relaxing of the definition of inner product after Proposition 3.5, the theorem suggests the feature mapping

$$\mathbf{x} = (x_1,\ldots,x_n) \longmapsto \phi(\mathbf{x}) = (\phi_1(\mathbf{x}),\ldots,\phi_j(\mathbf{x}),\ldots),$$

into the Hilbert space defined by the weighted inner product given by

$$\left\langle \psi \cdot \tilde{\psi} \right\rangle = \sum_{j=1}^{\infty} \lambda_j \psi_j \tilde{\psi}_j,$$

since the inner product of two feature vectors then satisfies

$$\langle \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{z}) \rangle = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{x}) \phi_j(\mathbf{z}) = K(\mathbf{x}, \mathbf{z}),$$

These features have the special property that they are orthonormal functions in $L_2(X)$. This choice of mapping is sometimes referred to as *Mercer features*. The compactness of the input domain is required in order to ensure that the spectrum is a countable set. Note that we do not need the features to form an orthonormal set. In the finite input space example given above they have been rescaled by the square root of the eigenvalues. In general we can rescale each coordinate,

$$\mathbf{x} = (x_1, \ldots, x_n) \longmapsto \boldsymbol{\phi}(\mathbf{x}) = (b_1 \phi_1(\mathbf{x}), \ldots, b_j \phi_j(\mathbf{x}), \ldots),$$

into the Hilbert space defined by the weighted inner product given by

$$\left\langle \psi \cdot \tilde{\psi} \right\rangle = \sum_{j=1}^{\infty} \frac{\lambda_j}{b_j^2} \psi_j \tilde{\psi}_j,$$

since again the inner product of two feature vectors then satisfies

$$\langle \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{z}) \rangle = \sum_{j=1}^{\infty} \frac{\lambda_j}{b_j^2} b_j \phi_j(\mathbf{x}) b_j \phi_j(\mathbf{z}) = K(\mathbf{x}, \mathbf{z}).$$

In this case and in the finite input space example the features are still orthogonal. Orthogonality is, however, also not required. For instance in the case of polynomial kernels, the features given in Example 3.3 will not in general be orthogonal. The features could be chosen to be orthogonal, but the particular polynomials needed would depend on the subset of $\mathbb{R}^2$ taken as the input domain and the measure of integration chosen.

Mercer features provide a representation of the input points by means of their image in the feature space with the inner product defined by the potentially infinite number of eigenvalues of the kernel. The sub-manifold formed by the image of the input space is defined by the eigenvectors of the kernel operator. Although the images of the points $(\mathbf{x})$ F will not be computed, their inner products can be calculated using the kernel function.

Example 3.9

Consider the kernel function $K(\mathbf{x},\mathbf{z}) = K(\mathbf{x}-\mathbf{z})$. Such a kernel is said to be translation invariant, since the inner product of two inputs is unchanged if both are translated by the same vector. Consider the one dimensional case in which K is defined on the interval $[0, 2\ ]$ in such a way that $K(u)$ can be extended to a continuous, symmetric, periodic function on $\mathbb{R}$. Such a function can be expanded in a uniformly convergent Fourier series:

$$K(u) = \sum_{n=0}^{\infty} a_n \cos(nu).$$

In this case we can expand $K(x - z)$ as follows:

$$K(x-z) = a_0 + \sum_{n=1}^{\infty} a_n \sin(nx)\sin(nz) + \sum_{n=1}^{\infty} a_n \cos(nx)\cos(nz).$$

Provided the a_n are all positive this shows $K(x,z)$ as the inner product in the feature space defined by the orthogonal features

$$\{\phi_i(x)\}_{i=0}^{\infty} = (1, \sin(x), \cos(x), \sin(2x), \cos(2x), \ldots, \sin(nx), \cos(nx), \ldots),$$

since the functions, 1, cos(nu) and sin(nu) form a set of orthogonal functions on the interval [0, 2]. Hence, normalising them will provide a set of Mercer features. Note that the embedding is defined independently of the a_n, which subsequently control the geometry of the feature space.

Example 3.9 provides some useful insight into the role that the choice of kernel can play. The parameters a_n in the expansion of $K(u)$ are its Fourier coefficients. If for some n, we have $a_n = 0$, the corresponding features are removed from the feature space. Similarly, small values of a_n mean that the feature is given low weighting and so will have less influence on the choice of hyperplane. Hence, the choice of kernel can be seen as choosing a filter with a particular spectral characteristic, the effect of which is to control the influence of the different frequencies in determining the optimal separation. In the next subsection we introduce a view of the feature space that will elucidate its role in encoding a learning bias.

Given a feature space representation with countably many linearly independent features (not necessarily orthogonal or of unit norm) given by the mapping

$$\mathbf{x} = (x_1, \ldots, x_n) \longmapsto \boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_j(\mathbf{x}), \ldots),$$

into the l_2 space F defined by the weighted inner product

$$\left\langle \psi \cdot \tilde{\psi} \right\rangle = \sum_{j=1}^{\infty} \mu_j \psi_j \tilde{\psi}_j,$$

we can define a space of functions on the input space to be the image of F under the mapping

$$T : \psi \longmapsto \sum_{j=1}^{\infty} \psi_j \phi_j(\mathbf{x}). \qquad (3.1)$$

Note that if F is finite dimensional, is the function class on the input space we are effectively using by applying linear functions in the feature space since it is the set of all linear combinations of the basis functions. For infinite feature spaces, may not contain all the possible hypothesis functions, as they may be images of points that do not have a finite norm in F, or equally may have too many functions. In the next subsection we consider a particular choice of the feature mapping that ensures does contain exactly the set of hypotheses and at the same time has a number of additional special properties.

Reproducing Kernel Hilbert Spaces

Assume we have a feature space given by the map

$$\mathbf{x} = (x_1, \ldots, x_n) \longmapsto \boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_j(\mathbf{x}), \ldots),$$

into the l_2 space F defined by the weighted inner product

$$\left\langle \psi \cdot \tilde{\psi} \right\rangle = \sum_{j=1}^{\infty} \mu_j \psi_j \tilde{\psi}_j,$$

where

$$K(\mathbf{x},\mathbf{z}) = \sum_{i=1}^{\infty} \mu_i \phi_i(\mathbf{x}) \phi_i(\mathbf{z}).$$

Now consider introducing a weighting proportional to the factor μ_i, so that the image has the form

$$\mathbf{x} = (x_1, \ldots, x_n) \longmapsto \boldsymbol{\psi}(\mathbf{x}) = (\psi_1(\mathbf{x}), \ldots, \psi_j(\mathbf{x}), \ldots) = (\mu_1 \phi_1(\mathbf{x}), \ldots, \mu_j \phi_j(\mathbf{x}), \ldots).$$

The appropriate weighted inner product is then given by

$$\left\langle \boldsymbol{\psi} \cdot \tilde{\boldsymbol{\psi}} \right\rangle_{\mu} = \sum_{j=1}^{\infty} \frac{\psi_j \tilde{\psi}_j}{\mu_j},$$

so that

$$\|\boldsymbol{\psi}(\mathbf{x})\|_{\mu}^2 = K(\mathbf{x},\mathbf{x}).$$

We denote with the image of F under the mapping T defined by equation (3.1). This particular weighting has a number of special properties. For two functions

$$f(\mathbf{x}) = \sum_{j=1}^{\infty} \psi_j \phi_j(\mathbf{x}) \text{ and } g(\mathbf{x}) = \sum_{j=1}^{\infty} \tilde{\psi}_j \phi_j(\mathbf{x}),$$

we define an inner product in by

$$\langle f(\cdot) \cdot g(\cdot) \rangle_{\mathcal{H}} = \sum_{j=1}^{\infty} \frac{\psi_j \tilde{\psi}_j}{\mu_j}$$

hence making the map T an isometry. First observe that if we map an input point to the feature space and then apply T to its image, we obtain

$$T(\boldsymbol{\psi}(\mathbf{z})) = \sum_{j=1}^{\infty} \psi_j(\mathbf{z}) \phi_j(\mathbf{x}) = \sum_{j=1}^{\infty} \mu_j \phi_j(\mathbf{z}) \phi_j(\mathbf{x}) = K(\mathbf{z},\mathbf{x}),$$

so that $K(\mathbf{z}, \cdot)$. As a consequence functions in the dual representation

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x})$$

are in the space . Furthermore, if we take the inner product of a general function f(**x**) Σ$^{∞}_{j=1}$ a_j $_j$(**x**) with K(**z**,**x**), we obtain

$$\begin{aligned} \langle f(\cdot) \cdot K(\mathbf{z},\cdot) \rangle_{\mathcal{H}} &= \left\langle (a_j)_{j=1}^{\infty} \cdot (\mu_j \phi_j(\mathbf{z}))_{j=1}^{\infty} \right\rangle_{\mu} \\ &= \sum_{j=1}^{\infty} \frac{a_j \mu_j \phi_j(\mathbf{z})}{\mu_j} = \sum_{j=1}^{\infty} a_j \phi_j(\mathbf{z}) = f(\mathbf{z}), \end{aligned}$$

known as the reproducing property of the kernel K. This also implies that coincides with the closure of the subspace

$$\mathcal{X} = \left\{ \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}) : \ell \in \mathbb{N},\ (\mathbf{x}_1, \ldots, \mathbf{x}_\ell) \in X^{\ell},\ \alpha_i \in \mathbb{R} \right\},$$

since if f satisfies f ⊣ 퓚 then for all **z** X,

$$f(\mathbf{z}) = \langle f(\cdot) \cdot K(\mathbf{z},\cdot) \rangle_{\mathcal{H}} = 0,$$

implying $f = 0$. Hence is contained in the closure of and so does not contain functions that cannot be arbitrarily well approximated in the dual representation. For two functions $f(\mathbf{x}) = \sum_{i=1}$ $_i K(\mathbf{x}_i, \mathbf{x})$ and $g(\mathbf{x}) = \sum_{j=1}^{\hat{\ell}} \hat{\alpha}_j K(\hat{\mathbf{x}}_j, \mathbf{x})$, in the dual representation the inner product is given by

$$\begin{aligned}
\langle f(\cdot) \cdot g(\cdot) \rangle_{\mathcal{H}} &= \left\langle \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \cdot) \cdot \sum_{j=1}^{\hat{\ell}} \hat{\alpha}_j K(\hat{\mathbf{x}}_j, \cdot) \right\rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{\ell} \alpha_i \sum_{j=1}^{\hat{\ell}} \hat{\alpha}_j \langle K(\mathbf{x}_i, \cdot) \cdot K(\hat{\mathbf{x}}_j, \cdot) \rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{\ell} \alpha_i \sum_{j=1}^{\hat{\ell}} \hat{\alpha}_j K(\mathbf{x}_i, \hat{\mathbf{x}}_j) \\
&= \sum_{i=1}^{\ell} \alpha_i g(\mathbf{x}_i) = \sum_{j=1}^{\hat{\ell}} \hat{\alpha}_j f(\hat{\mathbf{x}}_j), \qquad (3.2)
\end{aligned}$$

showing that the definition of the inner product is independent of the particular representation of the function (changing the representation of g does not change the value of $g(\mathbf{x}_i)$). In addition we also obtain that $\|f\|^2$ = Σ $_{i=1}$ $_i$f(x$_i$), showing that for f to have a bounded norm, it must have bounded value and coefficients. Finally, note that the reproducing property $f(\cdot) \cdot K(z, \cdot)$ $= f(z)$ implies that the evaluation functionals defined by $F_x[f] = f(x)$ (∀f) are linear and bounded, that is there exist $U_z = \|K(z,\cdot)\|$ $\mathbb{R}^+$ such that by the Cauchy-Schwarz inequality

$$|F_{\mathbf{z}}[f]| = |f(\mathbf{z})| = \langle f(\cdot) \cdot K(\mathbf{z}, \cdot) \rangle_{\mathcal{H}} \leq U_{\mathbf{z}} \|f\|_{\mathcal{H}}$$

for all f .

For a Hilbert space of functions defined over the input domain X ⊂ $\mathbb{R}^d$, the bounded linearity of the evaluation functionals is the defining property for a *reproducing kernel Hilbert space (RKHS)*. Hence, we have demonstrated the following result.

Theorem 3.10

For every Mercer kernel $K(\mathbf{x},\mathbf{z})$ defined over the domain X ⊂ $\mathbb{R}^d$, there exists an RKHS of functions defined over X for which K is the reproducing kernel.

Remarkably the converse of this theorem also holds. That is, for any Hilbert space of functions in which the evaluation functionals are bounded and linear, there exists a reproducing kernel function. That a reproducing kernel is also a Mercer kernel follows from the fact that for $f(\mathbf{x}) = \sum_{i=1}$ $_i K(\mathbf{x}_i, \mathbf{x})$

$$\begin{aligned}
0 \leq \|f\|_{\mathcal{H}}^2 &= \left\langle \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \cdot) \cdot \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \cdot) \right\rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{\ell} \alpha_i \sum_{j=1}^{\ell} \alpha_j \langle K(\mathbf{x}_i, \cdot) \cdot K(\mathbf{x}_j, \cdot) \rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j),
\end{aligned}$$

implying K is positive semi-definite for every finite subset of X, sufficient to imply the positivity of the operator K by Remark 3.7.

The following example suggests some of the power and insight that the RKHS construction affords.

Example 3.11

Suppose we wish to perform regression based on as set of training points

$$S = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_\ell, y_\ell)) \subset (X \times Y)^\ell \subset (\mathbb{R}^n \times \mathbb{R})^\ell,$$

generated from the target function $t(\mathbf{x})$. If we assume a dual representation of the form

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}),$$

we can seek to minimise the norm

$$\begin{aligned}
\|f - t\|_{\mathcal{H}}^2 &= \left\langle \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}) - t(\mathbf{x}) \cdot \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}) - t(\mathbf{x}) \right\rangle_{\mathcal{H}} \\
&= -2 \left\langle t(\mathbf{x}) \cdot \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}) \right\rangle_{\mathcal{H}} + \|f\|_{\mathcal{H}}^2 + \|t\|_{\mathcal{H}}^2 \\
&= -2 \sum_{i=1}^{\ell} \alpha_i \left\langle t(\mathbf{x}) \cdot K(\mathbf{x}_i, \mathbf{x}) \right\rangle_{\mathcal{H}} + \|f\|_{\mathcal{H}}^2 + \|t\|_{\mathcal{H}}^2 \\
&= -2 \sum_{i=1}^{\ell} \alpha_i y_i + \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) + \|t\|_{\mathcal{H}}^2,
\end{aligned}$$

which can be solved by setting the derivatives with respect to the parameters equal to zero, since $\|t\|^2$ does not depend on them. We will discuss this further in Chapter 5 in Example 5.6. Notice again how the Gram matrix in the feature space appears if we view K as a Mercer kernel.

3.3.2 Making Kernels from Kernels

The key to verifying that a new symmetric function is a kernel will be the conditions outlined in Remark 3.7, that is the requirement that the matrix defined by restricting the function to any finite set of points is positive semi-definite. We apply this criterion to confirm that a number of new kernels can be created. The following proposition can be viewed as showing that kernels satisfy a number of closure properties, allowing us to create more complicated kernels from simple building blocks.

Proposition 3.12

Let K_1 and K_2 be kernels over $X \times X$, X $\mathbb{R}$″, a $\mathbb{R}^+$, $f(\cdot)$ a real-valued function on X,

$$\phi : X \longrightarrow \mathbb{R}^m$$

with K_3 a kernel over $\mathbb{R}^m \times \mathbb{R}^m$, and $\mathbf{B}$ a symmetric positive semi-definite $n \times n$ matrix. Then the following functions are kernels:

1. $K(\mathbf{x},\mathbf{z}) = K_1(x,z) + K_2(\mathbf{x},\mathbf{z})$,
2. $K(\mathbf{x},\mathbf{z}) = aK_1(\mathbf{x},\mathbf{z})$,
3. $K(\mathbf{x},\mathbf{z}) = K_1(x,z)K_2(\mathbf{x},\mathbf{z})$,
4. $K(\mathbf{x},\mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$,
5. $K(\mathbf{x},\mathbf{z}) = K_3(\mathbf{phi;}(\mathbf{x}), (\mathbf{z}))$,
6. $K(\mathbf{x},\mathbf{z}) = \mathbf{x}\ \mathbf{Bz}$.

Proof Fix a finite set of points $\{\mathbf{x}_1,\ldots,\mathbf{x}\ \}$, and let $\mathbf{K}_1$ and $\mathbf{K}_2$, be the corresponding matrices obtained by restricting K_1 and K_2 to these points. Consider any vector $\quad\mathbb{R}$. Recall that a matrix $\mathbf{K}$ is positive semi-definite if and only if $\quad\mathbf{K}\quad\geq 0$, for all $\quad$.

1. We have

$$\alpha'(\mathbf{K}_1+\mathbf{K}_2)\alpha=\alpha'\mathbf{K}_1\alpha+\alpha'\mathbf{K}_2\alpha\geq 0,$$

   and so $\mathbf{K}_1 + \mathbf{K}_2$ is positive semi-definite and $K_1 + K_2$ a kernel function.
2. Similarly $\quad a\mathbf{K}_1\quad = a\quad\mathbf{K}_1\quad\geq 0$, verifying that aK_1 is a kernel.
3. Let

$$\mathbf{K}=\mathbf{K}_1\otimes\mathbf{K}_2$$

   be the tensor product of the matrices $\mathbf{K}_1$ and $\mathbf{K}_2$. The tensor product of two positive semi-definite matrices is positive semi-definite since the eigenvalues of the product are all pairs of products of the eigenvalues of the two components. The matrix corresponding to the function K_1K_2 is known as the Schur product $\mathbf{H}$ of $\mathbf{K}_1$ and $\mathbf{K}_2$ with entries the products of the corresponding entries in the two components. The matrix $\mathbf{H}$ is a principal submatrix of $\mathbf{K}$ defined by a set of columns and the same set of rows. Hence for any $\quad\mathbb{R}$, there is a corresponding $\ _1\quad\mathbb{R}^{\ 2}$, such that

$$\alpha'\mathbf{H}\alpha=\alpha_1'\mathbf{K}\alpha_1\geq 0,$$

   and so $\mathbf{H}$ is positive semi-definite as required.
4. We can rearrange the bilinear form as follows:

$$\begin{aligned}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\alpha_i\alpha_j K(\mathbf{x}_i,\mathbf{x}_j)&=\sum_{i=1}^{\ell}\sum_{j=1}^{\ell}\alpha_i\alpha_j f(\mathbf{x}_i)f(\mathbf{x}_j)\\&=\sum_{i=1}^{\ell}\alpha_i f(\mathbf{x}_i)\sum_{j=1}^{\ell}\alpha_j f(\mathbf{x}_j)\\&=\left(\sum_{i=1}^{\ell}\alpha_i f(\mathbf{x}_i)\right)^2\geq 0,\end{aligned}$$

   as required.
5. Since K_3 is a kernel, the matrix obtained by restricting K_3 to the points $\quad(\mathbf{x}_1),\ldots,\quad(\mathbf{x}\ )$ is positive semi-definite as required.
6. Consider the diagonalisation of $\mathbf{B} = \mathbf{V}\quad\mathbf{V}$ by an orthogonal matrix $\mathbf{V}$, where $\quad$ is the diagonal matrix containing the non-negative eigenvalues. Let $\sqrt{\Lambda}$ be the diagonal matrix with the square roots of the eigenvalues and set $\mathbf{A}=\sqrt{\Lambda}\mathbf{V}$. We therefore have

$$K(\mathbf{x},\mathbf{z})=\mathbf{x}'\mathbf{B}\mathbf{z}=\mathbf{x}'\mathbf{V}'\Lambda\mathbf{V}\mathbf{z}=\mathbf{x}'\mathbf{V}'\sqrt{\Lambda}\sqrt{\Lambda}\mathbf{V}\mathbf{z}=\mathbf{x}'\mathbf{A}'\mathbf{A}\mathbf{z}=\langle\mathbf{A}\mathbf{x}\cdot\mathbf{A}\mathbf{z}\rangle,$$

   the inner product using the feature mapping $\mathbf{A}$.

Corollary 3.13

Let $K_1(\mathbf{x},\mathbf{z})$ be a kernel over $X \times X$, $\mathbf{x}, \mathbf{z}\quad X$, and $p(x)$ a polynomial with positive coefficients. Then the following functions are also kernels:

1. $K(\mathbf{x},\mathbf{z}) = p(K_1(\mathbf{x},\mathbf{z}))$,
2. $K(\mathbf{x},\mathbf{z}) = \exp(K(\mathbf{x},\mathbf{z}))$,
3. $K(\mathbf{x},\mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2 / \ ^2)$.

Proof We take the three parts in turn:

1. For a polynomial the result is immediate by combining the parts of the proposition. Note that the constant is covered by part 4 of the proposition.
2. The exponential function can be arbitrarily closely approximated by polynomials with positive coefficients and hence is a limit of kernels. Since kernels are clearly closed under taking pointwise limits, the result follows.
3. We can decompose the Gaussian function as follows:

   $$\exp(-\|\mathbf{x}-\mathbf{z}\|^2/\sigma^2) = \exp(-\|\mathbf{x}\|^2/\sigma^2)\exp(-\|\mathbf{z}\|^2/\sigma^2)\exp(2\langle \mathbf{x}\cdot\mathbf{z}\rangle/\sigma^2).$$

   The first two factors together form a kernel by part 4 of the proposition, while the third factor is a kernel by part 2 of this corollary.

Remark 3.14

The final kernel of the corollary is known as the Gaussian kernel. This function forms the core of a radial basis function network and hence, using this kernel will mean the hypotheses are radial basis function networks.

3.3.3 Making Kernels from Features

Another way to obtain a kernel is of course to start from the features, and to obtain it by working out their inner product. In this case there is no need to check for positive semi-definiteness since this will follow automatically from the definition as an inner product. The first example we gave of polynomial kernels followed this route. We now give a more unusual example of a kernel over a discrete space, in this case finite strings in order to illustrate the potential of the method for non-Euclidean spaces.

Example 3.15

(*String subsequence kernels*) Let Σ be a finite alphabet. A string is a finite sequence of characters from Σ, including the empty sequence. For strings *s, t*, we denote by $|s|$ the length of the string $s = s_1 \ldots s_{|s|}$, and by *st* the string obtained by concatenating the strings *s* and *t*. The string $s[i:j]$ is the substring $s_i \ldots s_j$ of *s*. We say that *u* is a subsequence of *s*, if there exist indices $\mathbf{i} = (i_1,\ldots,i_{|u|})$, with $1 \le i_1 < \ldots < i_{|u|} \le |s|$, such that $u_j = s_{ij}$, for $j = 1,\ldots,|u|$, or $u = s[\mathbf{i}]$ for short. The length $l(\mathbf{i})$ of the subsequence in *s* is $i_{|u|} - i_1 + 1$. We denote by Σ the set of all finite strings of length *n*, and by Σ∗ the set of all strings

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n.$$

We now define feature spaces $F_n = \mathbb{R}^{\Sigma^n}$. The feature mapping for a string *s* is given by defining the *u* coordinate $\phi_u(s)$ for each $u \in \Sigma^n$. We define

$$\phi_u(s) = \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})},$$

for some $\lambda \le 1$. These features measure the number of occurrences of subsequences in the string *s* weighting them according to their lengths. Hence, the inner product of the feature vectors for two strings *s* and *t* give a sum over all common subsequences weighted according to their frequency of occurrence and lengths

$$
\begin{aligned}
K_n(s,t) &= \sum_{u\in\Sigma^n} \langle \phi_u(s)\cdot\phi_u(t)\rangle \\
&= \sum_{u\in\Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(\mathbf{i})} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{j})} \\
&= \sum_{u\in\Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{i})+l(\mathbf{j})}.
\end{aligned}
$$

Such a kernel could clearly be very useful for text classification, but appears at first sight too expensive to compute for practical purposes. We introduce an additional function which will aid in defining a recursive computation for this kernel. Let

$$
K'_i(s,t) = \sum_{u\in\Sigma^i} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{|s|+|t|-i_1-j_1+2},\ i=1,\ldots,n-1,
$$

that is counting the length to the end of the strings s and t instead of just $l(\mathbf{i})$ and $l(\mathbf{j})$. We can now define a recursive computation for K'_i and hence compute K_n,

$$
\begin{aligned}
K'_0(s,t) &= 1, \text{ for all } s,t, \\
K'_i(s,t) &= 0, \text{ if } \min\left(|s|,|t|\right) < i, \\
K_i(s,t) &= 0, \text{ if } \min\left(|s|,|t|\right) < i, \\
K'_i(sx,t) &= \lambda K'_i(s,t) + \sum_{j:t_j=x} K'_{i-1}(s,t[1:j-1])\lambda^{|t|-j+2},\ i=1,\ldots,n-1, \\
K_n(sx,t) &= K_n(s,t) + \sum_{j:t_j=x} K'_{n-1}(s,t[1:j-1])\lambda^2.
\end{aligned}
$$

The correctness of this recursion follows from observing how the length of the strings has increased, incurring a factor of λ for each extra character, until the full length of n characters has been attained. It is quite surprising that the recursive formulation is sufficient to compute the kernel in time proportional to $n|s||t|$. If we wished to compute $K_n(s,t)$ for a range of values of n, we would simply perform the computation of $K'_i(s,t)$ up to one less than the largest n required, and then apply the last recursion for each $K_n(s,t)$ that is needed using the stored values of $K'_i(s,t)$. We can of course use parts 1 and 2 of Proposition 3.12 to create a kernel $K(s,t)$ that combines the different $K_n(s,t)$ giving different weightings for each n.

12 Chapter 3: Kernel-Induced Feature Spaces

3.4 Working in Feature Space

Since the embedding given by the feature mapping

$$\mathbf{x} = (x_1, \ldots, x_n) \longmapsto \boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_j(\mathbf{x}), \ldots)$$

is non-linear and hence in general defines an n dimensional sub-manifold of the feature space, linear combinations of images often do not correspond to images of any input point. Nevertheless, we are able to represent such points using the dual representation, and provided we are only concerned with distances and inner products the kernel can be used without computing an explicit feature vector. In this section, we briefly review how such calculations can be performed. Let (*X*) be the image of the input space under the feature mapping, and define *F* = co((*X*)) to be the space of all finite linear combinations of points from (*X*). We represent a generic point

$$P = \sum_{i=1}^{\ell} \alpha_i \boldsymbol{\phi}(\mathbf{x}_i)$$

in *F*, by a sequence of pairs

$$P = (\alpha_i, \mathbf{x}_i)_{i=1}^{\ell}.$$

Consider a second such point *Q* = (β_{*i*},**z**_{*ii*=1}. The sum and difference of two such points are defined by taking the union of the set of points and adding or subtracting corresponding coefficients. For example if $\mathbf{x}_i$, *i* = 1,…, , are a set of positive examples, and $\mathbf{z}_i$, *i* = 1,…,*s*, a set of negative examples, then the weight vector of the hyperplane oriented from the centre of mass of the negative points towards the centre of mass of the positives is given by

$$\frac{1}{\ell}\sum_{i=1}^{\ell}(1, \mathbf{x}_i) - \frac{1}{s}\sum_{j=1}^{s}(1, \mathbf{z}_j) = \left(\frac{1}{\ell}, \mathbf{x}_i\right)_{i=1}^{\ell} \cup \left(-\frac{1}{s}, \mathbf{z}_j\right)_{j=1}^{s}.$$

The inner product with a second point *Q* = (β_{*i*},$\mathbf{z}_i$)^s_{*i*=1} is given by

$$\langle P \cdot Q \rangle_F = \sum_{i,j} \alpha_i \beta_j K(\mathbf{x}_i, \mathbf{z}_j).$$

Note that this corresponds exactly to the inner product $f \cdot g$ of the two functions

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}) \quad \text{and} \quad g(\mathbf{x}) = \sum_{i=1}^{s} \beta_i K(\mathbf{z}_i, \mathbf{x}),$$

in the corresponding RKHS . Hence, we can also view the sequences of pairs as representing functions in the RKHS with the inner product giving the corresponding RKHS inner product.

For a point **x** *X* the norm squared of the feature vector is *K* (**x,x**). Note that for Gaussian kernels, this is equal to 1 for all inputs, showing that the input space is embedded onto the surface of the unit ball in this case. Similarly, the square of the distance between *P* and *Q* can be computed as

$$\begin{aligned} \|P - Q\|_F^2 &= \langle P - Q \cdot P - Q \rangle_F \\ &= \sum_{i,j} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - 2\sum_{i,j} \alpha_i \beta_j K(\mathbf{x}_i, \mathbf{z}_j) + \sum_{i,j} \beta_i \beta_j K(\mathbf{z}_i, \mathbf{z}_j). \end{aligned}$$

As an example of how distances can be computed, the conditions that *P* must satisfy to be at the centre of the smallest sphere containing the images (1,$\mathbf{x}_i$) of the points $\mathbf{x}_i$, *i* = 1,…, , are

$$P = (\alpha_i, \mathbf{x}_i)_{i=1}^{\ell}, \text{ where}$$

$$\boldsymbol{\alpha} = \underset{\boldsymbol{\alpha}}{\operatorname{argmin}} \left(\sum_{i,j=1}^{\ell} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) + \max_{1 \le k \le \ell} \left(K(\mathbf{x}_k, \mathbf{x}_k) - 2 \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}_k) \right) \right),$$

since the squared distance from P to $(1, \mathbf{x}_k)$ is

$$\langle P - (1, \mathbf{x}_k) \cdot P - (1, \mathbf{x}_k) \rangle_F = \sum_{i,j=1}^{\ell} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - 2 \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}_k) + K(\mathbf{x}_k, \mathbf{x}_k),$$

and the first term on the right hand side is independent of k.

Following this type of approach it is also possible to perform principal components analysis in feature space using dual representations of the feature vectors.

Remark 3.16

There is an interesting view of the feature space, not necessary for the main development of the book, that sheds light on the structure of the image manifold of the input space. The distances we have considered above are all distances in the full feature space, which is potentially very high dimensional. The image $\Phi(X)$ of the input space is a possibly contorted submanifold whose dimension is that of the input space. It is also possible to analyse distances within this image. The measurement of distance along the surface requires the use of a Riemannian metric defined by an appropriate tensor. Differential geometry studies the metric induced on a space by such a tensor. Metric tensors are symmetric and positive definite. Choosing a kernel induces a corresponding tensor g, which can be determined by computing the first three terms of a Taylor series for the squared distance between two points $\mathbf{x}$ and $\mathbf{z}$ as a function of the point $\mathbf{x} = \mathbf{z} + d\mathbf{x}$ on the surface. Since the first two terms are zero we obtain the bilinear form or quadratic term

$$\begin{aligned} ds^2 &= \langle (1, \mathbf{x}) - (1, \mathbf{z}) \cdot (1, \mathbf{x}) - (1, \mathbf{z}) \rangle_F \\ &= K(\mathbf{x}, \mathbf{x}) - 2K(\mathbf{x}, \mathbf{z}) + K(\mathbf{z}, \mathbf{z}) \\ &= \frac{1}{2} \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \left(\frac{\partial^2 K(\mathbf{x}, \mathbf{x})}{\partial x_i \partial x_j} \right)_{\mathbf{x}=\mathbf{z}} dx_i dx_j - \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \left(\frac{\partial^2 K(\mathbf{x}, \mathbf{z})}{\partial x_i \partial x_j} \right)_{\mathbf{x}=\mathbf{z}} dx_i dx_j \\ &= \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} \left(\frac{1}{2} \frac{\partial^2 K(\mathbf{x}, \mathbf{x})}{\partial x_i \partial x_j} - \frac{\partial^2 K(\mathbf{x}, \mathbf{z})}{\partial x_i \partial x_j} \right)_{\mathbf{x}=\mathbf{z}} dx_i dx_j, \end{aligned}$$

giving the tensor $\mathbf{g}$ components

$$g_{ij}(\mathbf{z}) = \left(\frac{1}{2} \frac{\partial^2 K(\mathbf{x}, \mathbf{x})}{\partial x_i \partial x_j} - \frac{\partial^2 K(\mathbf{x}, \mathbf{z})}{\partial x_i \partial x_j} \right)_{\mathbf{x}=\mathbf{z}}.$$

Hence, the Riemannian metric tensor can also be computed from the kernel function and may give additional insight into the structure of the feature space and the sub-manifold corresponding to the image of the input space.

3.5 Kernels and Gaussian Processes

If we consider the output of a function $f(\mathbf{x})$, for fixed $\mathbf{x} \in X$, as f is chosen according to some distribution 퓓 defined over a class of real-valued functions , we may view the output value as a random variable, and hence

$$\{f(\mathbf{x}) : \mathbf{x} \in X\}$$

as a collection of potentially correlated random variables. Such a collection is known as a stochastic process. The distribution over the function class can be regarded as our prior belief in the likelihood that the different functions will provide the solution to our learning problem. Such a prior is characteristic of a Bayesian perspective on learning. We will return to discuss this approach further in the next chapter and in Chapter 6 discuss how to make predictions using Gaussian processes. At this point we wish to highlight the connection between a particular form of prior commonly used in Bayesian learning and the kernel functions we have introduced for Support Vector Machines. Many of the computations required by a Bayesian analysis are greatly simplified if the prior distributions are assumed to be Gaussian distributions. For a finite set of variables $S = (\mathbf{x}_1,\ldots,\mathbf{x}_\ell)$, a Gaussian distribution (with zero mean) is specified by a symmetric positive definite covariance matrix Σ = Σ$(\mathbf{x}_1,\ldots,\mathbf{x}_\ell)$ with the corresponding distribution given by

$$P_{f\sim\mathcal{D}}[(f(\mathbf{x}_1),\ldots,f(\mathbf{x}_\ell)) = (y_1,\ldots,y_\ell)] \propto \exp\left(-\frac{1}{2}\mathbf{y}'\Sigma^{-1}\mathbf{y}\right).$$

A Gaussian process is a stochastic process for which the marginal distribution for any finite set of variables is zero mean Gaussian. The (i,j) entry of Σ measures the correlation between $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$, that is the expectation $E_{f\sim D}[f(\mathrm{x_i})f(\mathrm{x_j})]$, and hence depends only on $\mathbf{x}_i$ and $\mathbf{x}_j$. There therefore exists a symmetric covariance function $K(\mathbf{x},\mathbf{z})$ such that Σ$(\mathbf{x}_1,\ldots,\mathbf{x}_\ell)_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$. The requirement that the covariance matrix is positive definite for all finite sets of input points is precisely the defining property of a Mercer kernel given in Remark 3.7, and hence we see that defining a Gaussian process over a set of variables indexed by a space X is equivalent to defining a Mercer kernel on $X \times X$. The definition of a Gaussian process by specifying the covariance function avoids explicit definition of the function class , and the prior over the functions in . Hence, in much the same way that the feature space is defined implicitly by the kernel in Support Vector Machines, the function class and prior are defined implicitly by the Gaussian process covariance function. It is possible to define a function class and prior for which the kernel is the corresponding covariance function in much the same way that the feature space can be explicitly computed for a kernel. Indeed one choice of function space is the class of linear functions in the space F of Mercer features

$$\mathbf{x} = (x_1,\ldots,x_n) \longmapsto \boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}),\ldots,\phi_j(\mathbf{x}),\ldots),$$

in the l_2 space defined by the weighted inner product given by

$$\left\langle \psi \cdot \tilde{\psi} \right\rangle = \sum_{j=0}^{\infty} \lambda_j \psi_j \tilde{\psi}_j.$$

The prior distribution 퓓 over the weight vector ψ is chosen to be an independent zero mean Gaussian in each coordinate with variance in coordinate i equal to $\sqrt{\lambda_i}$. With this prior we can compute the correlation $C(\mathbf{x},\mathbf{z})$ between the outputs of two inputs $\mathbf{x}$ and $\mathbf{z}$, as follows:

$$
\begin{aligned}
C(\mathbf{x},\mathbf{z}) &= E_{\psi\sim\mathscr{D}}\left[\langle\boldsymbol{\psi}\cdot\boldsymbol{\phi}(\mathbf{x})\rangle\,\langle\boldsymbol{\psi}\cdot\boldsymbol{\phi}(\mathbf{z})\rangle\right] \\
&= \int_{\mathscr{F}} \langle\boldsymbol{\psi}\cdot\boldsymbol{\phi}(\mathbf{x})\rangle\,\langle\boldsymbol{\psi}\cdot\boldsymbol{\phi}(\mathbf{z})\rangle\, d\mathscr{D}(\boldsymbol{\psi}) \\
&= \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} \phi_i(\mathbf{x})\phi_j(\mathbf{z}) \int_{\mathscr{F}} \psi_i\psi_j d\mathscr{D}(\boldsymbol{\psi}) \\
&= \sum_{i=0}^{\infty}\sum_{j=0}^{\infty} \phi_i(\mathbf{x})\phi_j(\mathbf{z})\delta_{ij}\lambda_i \\
&= \sum_{i=0}^{\infty} \phi_i(\mathbf{x})\phi_i(\mathbf{z})\lambda_i = K(\mathbf{x},\mathbf{z}),
\end{aligned}
$$

as promised.

3.6 Exercises

1. Working in feature space, find the centre of mass and the mean of the squared distances of the points from it.
2. Let $K(\mathbf{x},\mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2 / \sigma^2)$ be the Gaussian kernel of Remark 3.14, which can be applied in any Euclidean or l_2 space. Now consider any kernel $K_1(\mathbf{x},\mathbf{z})$ over $X \times X$ for an input space X. Show how you can compute a Gaussian kernel of the features defined implicitly by K_1 and hence use it as a kernel over $X \times X$.
3. Consider Example 3.15. Using the same feature space construct a kernel for the feature map for which $\phi_u(s)$ counts the number of occurrences of u as a substring of s.

2 Chapter 3: Kernel-Induced Feature Spaces

3.7 Further Reading and Advanced Topics

The use of Mercer's theorem for interpreting kernels as inner products in a feature space was introduced into machine learning in 1964 by the work of Aizermann, Bravermann and Rozoener on the method of potential functions [1], but its possibilities were not fully understood until it was first used in the article by Boser, Guyon and Vapnik that introduced the Support Vector method [19].

The theory of kernels is, however, older: Mercer's theorem dates back to 1909 [95], and the study of reproducing kernel Hilbert spaces was developed by Aronszajn in the 1940s [7]. This theory was used in approximation and regularisation theory, see for example the book of Wahba [171] and her 1999 survey [172]. The first use of polynomial kernels was by Poggio in 1975 [115]. Reproducing kernels were extensively used in machine learning and neural networks by Poggio and Girosi, see for example their 1990 paper on radial basis function networks [116].

The theory of positive definite functions was also developed in the context of covariance and correlation functions, so that work in Gaussian processes is closely related [180]. In fact that literature builds on the older results in [172]. Saitoh [123] shows the connection between positivity and the positive semi-definiteness of all finite set kernel matrices mentioned in Remark 3.7.

Techniques for ‘making kernels' can be found in many papers, for example by Micchelli [97], MacKay [81], Evgeniou et al. [39], Schölkopf et al. [136], Haussler [58], and Watkins [174]. The discussion about RKHSs draws on the paper of Haussler [58], while Example 3.15 is based on Watkins's paper [176]. The one dimensional shift invariant kernels of Example 3.9 is taken from Girosi [51]. The differential geometric description of the feature space has been provided by Burges [132], along with some necessary conditions for a kernel to satisfy Mercer's theorem.

Building on an observation of Schölkopf [129], Watkins [175] and Haussler [58] have greatly extended the use of kernels, showing that they can in fact be defined on general sets, which do not need to be Euclidean spaces, paving the way for their use in a swathe of new real-world applications, on input spaces as diverse as biological sequences, text, and images. These kernels generalise the idea of recursive ANOVA kernels described in Vapnik [159].

Joachims [67] and Dumais et al. [36] used sparse vectors to encode text features. Jaakkola and Haussler proposed to use a hidden Markov model in order to evaluate a kernel between biosequences [65], where the feature vector is the Fisher score of the distribution. This is described in more detail in Subsection 8.4.1. Watkins proposed to use probabilistic context free grammars to build kernels between sequences [174]. Also Haussler proposed the use of special kernels for biosequences [58].

An interesting direction of research is to learn the kernel directly from the data. The papers of Jaakkola [65] and Cristianini et al. [31] deal with this issue. An interesting paper by Amari and Wu [3] describes a method for directly acting on the kernel in order to affect the geometry in the input space, so that the separability of the data is improved.

The use of kernels as a general technique for using linear machines in a non-linear fashion can be *exported* to other learning systems, such as nearest neighbour (using the techniques of Section 3.4) or less trivially to PCA as demonstrated by Schölkopf, Smola and Mueller [134]. A technique that uses a specific kernel in order to deal with noisy and non-separable data was introduced by Shawe-Taylor and Cristianini, and will be described in Chapter 6 (see also [140]).

These references are also given on the website http://www.support-vector.net. which will be kept up to date with new work, pointers to software and papers that are available on-line.

2 Chapter 3: Kernel-Induced Feature Spaces

Chapter 4: Generalisation Theory

Overview

The introduction of kernels greatly increases the expressive power of the learning machines while retaining the underlying linearity that will ensure that learning remains tractable. The increased flexibility, however, increases the risk of overfitting as the choice of separating hyperplane becomes increasingly ill-posed due to the number of degrees of freedom.

In Chapter 1 we made several references to the reliability of the statistical inferences inherent in the learning methodology. Successfully controlling the increased flexibility of kernel-induced feature spaces requires a sophisticated theory of generalisation, which is able to precisely describe which factors have to be controlled in the learning machine in order to guarantee good generalisation. Several learning theories exist that can be applied to this problem. The theory of Vapnik and Chervonenkis (VC) is the most appropriate to describe SVMs, and historically it has motivated them, but it is also possible to give a Bayesian interpretation, among others.

In this chapter we review the main results of VC theory that place reliable bounds on the generalisation of linear classifiers and hence indicate how to control the complexity of linear functions in kernel spaces. Also, we briefly review results from Bayesian statistics and compression schemes that can also be used to describe such systems and to suggest which parameters to control in order to improve generalisation.

4.1 Probably Approximately Correct Learning

The model we will now introduce is known under a number of different names depending on the discipline concerned. Within statistics it would be known as the study of rates of uniform convergence, or frequentist inference, but within computer science it is generally referred to as the probably approximately correct or *pac* model, although Vapnik and Chervonenkis applied this style of analysis to statistical inference many years before it became popular in machine learning. The reason for the name will become apparent when we describe the components of the model.

The key assumption on which the model is based is that the data used in training and testing are generated independently and identically (i.i.d.) according to an unknown but fixed distribution 퓓. We assume this is a distribution over input/output pairings (**x**, y) $X \times \{-1, 1\}$, an approach which subsumes the case where the output y is determined by a fixed *target* function t of the input $y = t(\mathbf{x})$. Adaptations of the model have considered the case where the distribution changes over time, or where there is not full independence in the generation of the examples in the training set, as might be expected in for instance a sequence of examples generated as a time series. The model also ignores the possibility that the learner might be able to influence which examples are chosen, an ingredient that is studied in the query model of learning. We will ignore all these refinements and consider only the i.i.d. case.

Since the test examples are generated according to the distribution 퓓, the natural measure of error in the classification case is the probability that a randomly generated example is misclassified. Again consideration can be made of unequal costs for misclassification of positive and negative examples, but this question will be ignored in our initial analysis. We therefore define the error $\mathrm{err}_{\&\#54483;}(h)$ of a classification function h in distribution 퓓 to be

$$\operatorname*{err}_{\mathscr{D}}(h) = \mathscr{D}\{(\mathbf{x}, y) : h(\mathbf{x}) \neq y\}.$$

Such a measure is also referred to as a *risk functional*, as its measure the expected error rate. The aim of the analysis will be to assert bounds on this error in terms of several quantities. Perhaps the most crucial is the number of training examples used. Frequently pac results have been presented as bounds on the number of examples required to obtain a particular level of error. This is also known as the *sample complexity* of the learning problem. We prefer bounding the error in terms of the number of examples as this error can then be used directly as a criterion for choosing between different classes, the so-called model selection problem.

Consider a fixed inference rule for selecting a hypothesis h_S from the class H of classification rules at the learner's disposal based on a set

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$$

of training examples chosen i.i.d. according to 퓓. In this setting we can view the generalisation error $\mathrm{err}_{\&\#54483;}(h_S)$ as a random variable depending on the random selection of the training set. The statistical mechanical approach to analysing generalisation aims to bound the expected generalisation error, where the expectation is taken over the random selection of training sets of a particular size. There are situations where such an estimate can be unreliable, as a particular error may fall far from its expectation. An example of such an unreliable estimator is that given by cross-validation. The pac model of learning requires a generalisation error bound that is unlikely to fail. It therefore bounds the tail of the distribution of the generalisation error random variable $\mathrm{err}_{\&\#54483;}(h_S)$. The size of the tail is determined by a further parameter specified by the learner. Hence, a pac bound has the form = (, H,), and asserts that with probability at least 1 - over randomly generated training sets S, the generalisation error of the selected hypothesis h_S will be bounded by

$$\operatorname*{err}_{\mathscr{D}}(h_S) \leq \varepsilon(\ell, H, \delta), \qquad (4.1)$$

or in other words is *probably approximately correct (pac)*. This is equivalent to asserting that the probability that the training set gives rise to a hypothesis with large error is small:

$$\mathscr{D}^{\ell}\left\{S : \underset{\mathscr{D}}{\mathrm{err}}(h_S) > \varepsilon(\ell, H, \delta)\right\} < \delta. \qquad (4.2)$$

The pac approach has the flavour of a statistical test in the sense that it asserts that the probability the data have misled us is small. This corresponds to saying that a result is significant at the level, or in other words that the probability a test has misled us is at most . In this sense it provides a hypothesis with a statistical validation similar to those employed in developing the experimental sciences.

One of the key ingredients of the pac approach is that unlike many statistical tests the bound on the error should not depend on the distribution 퓓. This means that the bounds must hold whatever the distribution generating the examples, a property sometimes referred to as *distribution free*. It is not surprising that some distributions make learning harder than others, and so a theory that holds for all distributions must inevitably be pessimistic in many cases. We will see later that the large margin approach breaks this worst case deadlock and is able to take advantage of benign distributions. First, however, we will introduce the analysis of the distribution free case.

4.2 Vapnik Chervonenkis (VC) Theory

For a finite set of hypotheses it is not hard to obtain a bound in the form of inequality (4.1). Assume we use the inference rule that selects any hypothesis h that is consistent with the training examples in S. The probability that all of the independent examples are consistent with a hypothesis h for which $\text{err}_{퓓}(h) >$, is bounded by

$$\mathscr{D}^{\ell}\{S : h \text{ consistent and } \underset{\mathscr{D}}{\text{err}}(h) > \varepsilon\} \leq (1-\varepsilon)^{\ell} \leq \exp(-\varepsilon\ell),$$

where the second inequality is a simple mathematical bound. Now, even if we assume that all $|H|$ of the hypotheses have large error, the probability that one of them is consistent with S is at most

$$|H|\exp(-\varepsilon\ell),$$

by the union bound on the probability that one of several events occurs. This bounds the probability that a consistent hypothesis h_S has error greater than , as given in inequality (4.2),

$$\mathscr{D}^{\ell}\{S : h_S \text{ consistent and } \underset{\mathscr{D}}{\text{err}}(h_S) > \varepsilon\} < |H|\exp(-\varepsilon\ell).$$

In order to ensure the right hand side is less that , we set

$$\varepsilon = \varepsilon(\ell, H, \delta) = \frac{1}{\ell}\ln\frac{|H|}{\delta}.$$

This simple bound already shows how the complexity of the function class H measured here by its cardinality has a direct effect on the error bound. Clearly choosing H too large can lead to overfitting. The result also shows that a property relating the true error to the empirical error holds for all hypotheses in the set H. For this reason it is said to demonstrate *uniform convergence*. Learning theory relies on bounding the difference between empirical and true estimates of error uniformly over the set of hypotheses and conditions that can arise. As we have seen this is not difficult when the number of hypotheses is finite. The major contribution of the theory developed by Vapnik and Chervonenkis was to extend such an analysis to infinite sets of hypotheses, as for example in the case when we consider linear learning machines indexed by real-valued weight vectors.

We assume an inference rule that delivers any consistent hypothesis and denote by $\text{err}_S(h)$ the number of errors made by hypothesis h on the set S of examples. The key to bounding over an infinite set of functions is to bound the probability of inequality (4.2) by twice the probability of having zero error on the training examples but high error on a second random sample $\hat{S}$:

$$\mathscr{D}^{\ell}\left\{S : \exists h \in H : \underset{S}{\text{err}}(h) = 0, \underset{\mathscr{D}}{\text{err}}(h) > \varepsilon\right\}$$
$$\leq 2\mathscr{D}^{2\ell}\left\{S\hat{S} : \exists h \in H : \underset{S}{\text{err}}(h) = 0, \underset{\hat{S}}{\text{err}}(h) > \varepsilon/2\right\}. \qquad (4.3)$$

This relation follows from an application of Chernoff bounds provided > 2/ . The quantity on the right hand side is bounded by fixing the 2 sample and counting different orders in which the points might have been chosen while still keeping all the errors in the second sample. Since each order or permutation is equally likely the fraction of orders that satisfy the property is an upper bound on its probability. By only considering permutations that swap corresponding points from the first and second sample, we can bound the fraction of such permutations by 2^{-} $^{/2}$, independently of the particular set of 2 sample points. The advantage of considering errors over a finite set of 2 sample points is that the hypothesis space has effectively become finite, since there cannot be more than 2^{2} classification functions on 2 points. In order to obtain a union bound on the overall probability of the right hand side of inequality (4.3), all that is required is a bound on the size of the hypothesis space when restricted to 2 points, a quantity known as the *growth function*

$$B_H(\ell) = \max_{(\mathbf{x}_1,\ldots,\mathbf{x}_\ell)\in X^\ell} |\{(h(\mathbf{x}_1), h(\mathbf{x}_2), \ldots, h(\mathbf{x}_\ell)) : h \in H\}|,$$

where $|A|$ denotes the cardinality of the set A. The first observation about this quantity is that it cannot exceed 2^{ℓ} since the sets over which the maximum is sought are all subsets of the set of binary sequences of length ℓ. A set of points $\{\mathbf{x}_1,\ldots,\mathbf{x}_{\ell}\}$ for which the set

$$\{(h(\mathbf{x}_1), h(\mathbf{x}_2), \ldots, h(\mathbf{x}_{\ell})) : h \in H\} = \{-1, 1\}^{\ell}$$

is said to be *shattered* by H. If there are sets of any size which can be shattered then the growth_function is equal to 2^{ℓ} for all ℓ. The final ingredient in the Vapnik Chervonenkis theory is an analysis of the case when there is a finite d which is the largest size of shattered set. In this case the growth function can be bounded as follows for $\ell \geq d$

$$B_H(\ell) \leq \left(\frac{e\ell}{d}\right)^d,$$

giving polynomial growth with exponent d. The value d is known as the Vapnik Chervonenkis (VC) dimension of the class H, denoted by VCdim(H). These quantities measure the richness or flexibility of the function class, something that is also often referred to as its capacity. Controlling the capacity of a learning system is one way of improving its generalisation accuracy. Putting the above bound on the growth function together with the observation about the fraction of permutations for which the first half of the sample is able to mislead the learner, we obtain the following bound on the left hand side of inequality (4.2):

$$\mathscr{D}^{\ell}\left\{S : \exists h \in H : \underset{S}{\mathrm{err}}(h) = 0, \underset{\mathscr{D}}{\mathrm{err}}(h) > \varepsilon\right\} \leq 2\left(\frac{2e\ell}{d}\right)^d 2^{-\alpha\ell/2},$$

resulting in a pac bound for any consistent hypothesis h of

$$\underset{\mathscr{D}}{\mathrm{err}}(h) \leq \varepsilon(\ell, H, \delta) = \frac{2}{\ell}\left(d\log\frac{2e\ell}{d} + \log\frac{2}{\delta}\right),$$

where d = VCdim(H). Hence we have shown the following *fundamental theorem of learning*.

Theorem 4.1

(Vapnik and Chervonenkis) Let H be a hypothesis space having VC dimension d. For any probability distribution 퓓 on $X \times \{-1,1\}$, with probability 1 - over random examples S, any hypothesis h H that is consistent with S has error no more than

$$\underset{\mathscr{D}}{\mathrm{err}}(h) \leq \varepsilon(\ell, H, \delta) = \frac{2}{\ell}\left(d\log\frac{2e\ell}{d} + \log\frac{2}{\delta}\right),$$

provided $d \leq$ and $> 2/$.

Remark 4.2

The theorem shows that for infinite sets of hypotheses the problem of overfitting is avoidable and the measure of complexity that should be used is precisely the VC dimension. The size of training set required to ensure good generalisation scales linearly with this quantity in the case of a consistent hypothesis.

VC theory provides a distribution free bound on the generalisation of a consistent hypothesis, but more than that it can be shown that the bound is in fact tight up to log factors, as the following theorem makes clear.

Theorem 4.3

Let H be a hypothesis space with finite VC dimension $d \geq 1$. Then for any learning algorithm there exist distributions such that with probability at least over random examples, the error of the hypothesis h

returned by the algorithm is at least

$$\max\left(\frac{d-1}{32\ell}, \frac{1}{\ell}\ln\frac{1}{\delta}\right)$$

Remark 4.4

The theorem states that for a hypothesis class with high VC dimension there exist input probability distributions which will force the learner to require a large training set to obtain good generalisation. We can therefore see that finite VC dimension characterises learnability in the pac sense - we can bound the error as a function of a finite VCdim(H), while for unbounded VC dimension learning is impossible in the distribution free sense. Note, however, that the lower bound does not hold for all distributions. It is possible that a class with high VC dimension is learnable if the distribution is benign. Indeed this fact is essential for the performance of SVMs which are designed to take advantage of such benign distributions. This will be discussed further in thez next section.

In order to apply the theory to linear learning machines, we must compute the VCdim( ) of a linear function class in $\mathbb{R}^n$ in terms of the dimension n, that is determine what is the largest number d of examples that can be classified in all 2^d possible classifications by different linear functions, that is that can be shattered by . The following proposition characterises when this can be done.

Proposition 4.5

Let be the class of linear learning machines over $\mathbb{R}^n$.

1. Given any set S of $n + 1$ training examples in general position (not lying in an n - 1 dimensional affine subspace), there exists a function in that consistently classifies S, whatever the labelling of the training points in S.
2. For any set of $> n + 1$ inputs there is at least one classification that cannot be realised by any function in .

Theorem 4.3 and Proposition 4.5 imply that learning in very high dimensional feature spaces is not possible. An extreme example would be the use of a Gaussian kernel when we are effectively employing an infinite dimensional feature space. We must conclude that according to a distribution free pac analysis the Support Vector Machine approach to learning cannot succeed. The fact that SVMs can learn must therefore derive from the fact that the distribution generating the examples is not worst case as required for the lower bound of Theorem 4.3. In the next section we will sketch a more refined pac analysis that shows that the margin of a classifier provides a measure of how helpful the distribution is in identifying the target concept, resulting in a generalisation error bound of the form

$$\operatorname*{err}_{\mathcal{D}}(h) \leq \varepsilon(\ell, \mathcal{L}, \delta, \gamma)$$

that will not involve the dimension of the feature space. Hence, the SVM learning strategy is able to exploit collusions between distribution and target concept when they occur, as is frequently the case in real-world learning problems. Bounds of this type that involve quantities measured as a result of the training process will be referred to as *data dependent*.

The theory we have sketched so far only applies when the hypothesis is consistent with the training data. If there is noise in the data or the class of hypotheses is unable to capture the full richness of the target function, it may not be possible or advisable to aim for full consistency. The theory can be adapted to allow for a number of errors on the training set by counting the permutations which leave no more errors on the left hand side. The resulting bound on the generalisation error is given in the following theorem.

Theorem 4.6

Let H be a hypothesis space having VC dimension d. For any probability distribution 퓓 on $X \times \{-1, 1\}$, with probability 1 - over random examples S, any hypothesis h H that makes k errors on the training set S has error no more than

$$\underset{\mathscr{D}}{\mathrm{err}}(h) \leq \varepsilon(\ell, H, \delta) = \frac{2k}{\ell} + \frac{4}{\ell}\left(d \log \frac{2e\ell}{d} + \log \frac{4}{\delta}\right),$$

provided $d \leq$.

The theorem suggests that a learning algorithm for a hypothesis class H should seek to minimise the number of training errors, since everything else in the bound has been fixed by the choice of H. This inductive principle is known as empirical risk minimisation, since it seeks to minimise the empirically measured value of the risk functional. The theorem can also be applied to a nested sequence of hypothesis classes

$$H_1 \subset H_2 \subset \ldots \subset H_i \subset \ldots \subset H_M$$

by using /M, hence making the probability of any one of the bounds failing to hold to be less than . If a hypothesis h_i with minimum training error is sought in each class H_i, then the number of errors k_i that it makes on the fixed training set S will satisfy

$$k_1 \geq k_2 \geq \ldots \geq k_i \geq \ldots \geq k_M,$$

while the VC dimensions $d_i = \mathrm{VCdim}(H_i)$ form a non-decreasing sequence. The bound of Theorem 4.6 can be used to choose the hypothesis h_i for which the bound is minimal, that is the reduction in the number of errors (first term) outweighs the increase in capacity (second term). This induction strategy is known as structural risk minimisation.

4.3 Margin-Based Bounds on Generalisation

Recall that the definition of the margin of a classifier was given in Definition 2.2. We now generalise the definitions to an arbitrary class of real-valued functions.

Definition 4.7

Consider using a class of real-valued functions on an input space X for classification by thresholding at 0. We define the *margin of an example* $(\mathbf{x}_i, y_i)$ $X \times \{-1,1\}$ with respect to a function f to be the quantity

$$\gamma_i = y_i f(\mathbf{x}_i).$$

Note that $_i > 0$ implies correct classification of $(\mathbf{x}_i, y_i)$. The *margin distribution of f with respect to a training set S* is the distribution of the margins of the examples in S. We sometimes refer to the minimum of the margin distribution as the *margin* $m_S(f)$ *of f with respect to the training set S*. This quantity will be positive if f correctly classifies S. Finally, the *margin of a training set S* with respect to the class is the maximum margin over all f .

The following three subsections will consider bounds involving different measures of the margin distribution

$$M_S(f) = \{\gamma_i = y_i f(\mathbf{x}_i) : i = 1, \ldots, \ell\},$$

over a training set $S = ((\mathbf{x}_1, y_1),\ldots,(\mathbf{x} , y))$ for the real-valued function f. If we are considering a linear function class we assume that the margins are geometric (see Definition 2.2), or in other words that the weight vector has unit norm. We begin by considering the margin $m_S(f)$ or min $M_S(f)$.

4.3.1 Maximal Margin Bounds

When proving Theorem 4.1, we reduced the probability over an infinite set of hypotheses to the finite set of functions that can be realised on a 2 sample. A large margin can reduce the effective size of the function space still further because the generalisation performance can be approximated by a function whose output is within /2 on the points of the double sample. In many cases the size of a set of functions that approximate the behaviour of the whole class to within /2 on a fixed set of points is much smaller than the size of the growth function of the thresholded class. The estimation of the size of such a representative sample of functions requires some extra machinery and notation.

Definition 4.8

Let be a class of real-valued functions on a domain X. A *-cover* of with respect to a sequence of inputs

$$S = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_\ell)$$

is a finite set of functions B such that for all f , there exists g B, such that $\max_{1 \le i \le}$ ($|f(\mathbf{x}_i) - g(\mathbf{x}_i)|$) < . The size of the smallest such cover is denoted by 퓝(, S,), while the *covering numbers* of are the values

$$\mathcal{N}(\mathcal{F}, \ell, \gamma) = \max_{S \in X^\ell} \mathcal{N}(\mathcal{F}, S, \gamma).$$

We now show how for a hypothesis f with margin $m_S(f) =$ on the training set S, Theorem 4.1 can be reformulated in terms of the covering numbers of the underlying real-valued function class. We assume that there is a fixed threshold and that $\text{err}_S(f)$ counts the number of errors the thresholded output of f makes on the sample S. Similarly, for $\text{err}_{&\#54483;}(f)$ on a point generated randomly according to 퓓. We have

$$\mathscr{D}^{\ell}\left\{S : \exists f \in \mathscr{F} : \underset{S}{\mathrm{err}}(f) = 0, m_S(f) \geq \gamma, \underset{\mathscr{D}}{\mathrm{err}}(f) > \varepsilon\right\}$$
$$\leq 2\mathscr{D}^{2\ell}\left\{S\hat{S} : \exists f \in \mathscr{F} : \underset{S}{\mathrm{err}}(f) = 0, m_S(f) \geq \gamma, \underset{\hat{S}}{\mathrm{err}}(f) > \frac{\varepsilon\ell}{2}\right\}. \quad (4.4)$$

Consider a (/2)-cover *B* of with respect to the sequence *SŜ*. Let *g* *B*, be within /2 of *f*. It follows that *g* has $\mathrm{err}_S(g) = 0, m_S(g) >$ /2, while if *f* made an error on some point **x** Ŝ then *g* must have margin less than /2 on **x**. Hence, if (/2)-$\mathrm{err}_{\&\#348;}(g)$ denotes the number of points in *Ŝ* for which *g* has margin less that /2, we can bound the right hand side of inequality (4.4) by

$$2\mathscr{D}^{2\ell}\left\{S\hat{S} : \exists f \in \mathscr{F} : \underset{S}{\mathrm{err}}(f) = 0, m_S(f) \geq \gamma, \underset{\hat{S}}{\mathrm{err}}(f) > \varepsilon\ell/2\right\}$$
$$\leq 2\mathscr{D}^{2\ell}\left\{S\hat{S} : \exists g \in B : \underset{S}{\mathrm{err}}(g) = 0, m_S(g) > \gamma/2, (\gamma/2) - \underset{\hat{S}}{\mathrm{err}}(g) > \varepsilon\ell/2\right\}$$
$$\leq 2|B|2^{-\varepsilon\ell/2} \leq 2\mathscr{N}(\mathscr{F}, 2\ell, \gamma/2)2^{-\varepsilon\ell/2},$$

by a similar permutation argument and union bound. We have therefore demonstrated the following preliminary result.

Theorem 4.9

Consider thresholding a real-valued function space and fix $\mathbb{R}^+$. For any probability distribution 퓓 on $X \times \{-1, 1\}$, with probability 1 - over random examples *S*, any hypothesis *f* that has margin $m_S(f) \geq$ on *S* has error no more than

$$\underset{\mathscr{D}}{\mathrm{err}}(f) \leq \varepsilon(\ell, \mathscr{F}, \delta, \gamma) = \frac{2}{\ell}\left(\log \mathscr{N}(\mathscr{F}, 2\ell, \gamma/2) + \log\frac{2}{\delta}\right),$$

provided > 2/ .

Remark 4.10

The theorem shows how the generalisation error can be bounded in terms of the quantity $m_S(f)$ which is observed as a result of training. We expect that for larger values of , the size of log 퓝(, 2 , /2) will get smaller. This quantity can be viewed as an effective VC dimension and so we can expect that observing a large margin will result in good generalisation from a small sample. Notice that the VC dimension of does not enter into the bound. We will see examples later where the VC dimension is in fact infinite, but this effective VC dimension is still finite and hence learning can take place. This does not contradict the lower bound of Theorem 4.3, as the observation of a large margin indicates that the distribution is benign. Even though the bound holds for all distributions it will only be useful when the distribution is benign.

Remark 4.11

The theorem only applies for a fixed value of specified before the learning began. In order to be able to use the theorem for the observed value of after training, we must apply the theorem for a range of values, ensuring that there will be one close to any realistic outcome of training. The fact that the enters into the bound inside a log factor makes it possible to perform such a uniform application over a finite set without appreciable loss in the quality of the bound. The precise details of how we make the choice of different values of and obtain a uniform result become rather technical, but add little to the main message. We therefore will not go further into this question (see Section 4.8 for pointers to papers with more detail), but rather turn our attention to how we can bound log 퓝(, 2 , /2), the critical quantity if we want to make use of the result.

The bound on log 퓝(, ,) represents a generalisation of the bound on the growth function required for the VC theory. In that case the critical measure was the VC dimension *d* and the growth function was shown to grow polynomially with degree *d*. The corresponding quantity we shall use to bound the covering

numbers will be a real-valued generalisation of the VC dimension known as the fat-shattering dimension.

Definition 4.12

Let be a class of real-valued functions defined on a domain X. We say a set of points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}\}$ X is -*shattered* by , if there exist real numbers r_i, $i = 1, \ldots,$, such that for every binary classification $\mathbf{b}$ $\{-1, 1\}$, there exists $f_{\mathbf{b}}$, such that

$$f_{\mathbf{b}}(\mathbf{x}_i) \begin{cases} \geq r_i + \gamma, \text{ if } \mathbf{b}_i = 1, \\ < r_i - \gamma, \text{ if } \mathbf{b}_i = -1. \end{cases}$$

The *fat-shattering dimension* fat ( ) at scale is the size of the largest -shattered subset of X.

The dimension is also referred to as the scale-sensitive VC dimension. The real numbers r_i can be viewed as individual thresholds for each point while -shattering implies that we can realise every classification with margin relative to the chosen thresholds. Clearly the larger the value of , the smaller the size of set that can be shattered since the restrictions placed on the functions that can be used become stricter. If the thresholds r_i are required to be the same for all the points, the dimension is known as level fat-shattering. The freedom to choose individual thresholds appears to introduce extra flexibility, but in the case of linear functions it does not in fact increase the size of sets that can be shattered. We will return to the class of linear functions after considering the following bound on the covering numbers in terms of the fat-shattering dimension.

Lemma 4.13

Let be a class of functions X $[a, b]$ and 퓓 a distribution over X. Choose $0 <$ < 1 and let $d =$ fat (/4). Then for $\geq d$

$$\log \mathcal{N}(\mathcal{F}, \ell, \gamma) \leq 1 + d \log \frac{2e\ell(b-a)}{d\gamma} \log \frac{4\ell(b-a)^2}{\gamma^2}.$$

The bound on 퓝(, ,) is therefore slightly greater than polynomial, but if we ignore the log factors the dependency of log 퓝(, ,) on the fat-shattering dimension of exactly mimics the dependence of log B_H( ) on the VC dimension of H. We can therefore think of the fat-shattering dimension at scale /8 as the *effective* VC dimension when we observe a margin of . Indeed if we use Lemma 4.13 in Theorem 4.9 for a fixed value of , we obtain the following corollary.

Corollary 4.14

Consider thresholding a real-valued function space with range $[-R, R]$ and fix $\mathbb{R}^+$. For any probability distribution 퓓 on $X \times \{-1, 1\}$, with probability 1 - over random examples S, any hypothesis f that has margin $m_S(f) \geq$ on S has error no more than

$$\operatorname*{err}_{\mathcal{D}}(f) \leq \varepsilon(\ell, \mathcal{F}, \delta, \gamma) = \frac{2}{\ell}\left(d \log \frac{16e\ell R}{d\gamma} \log \frac{128\ell R^2}{\gamma^2} + \log \frac{4}{\delta}\right),$$

provided $> 2/$, $d <$, where $d =$ fat (/8).

Remark 4.15

Notice that if one ignores log factors, the role of the fat-shattering dimension in this bound is analogous to that of the VC dimension in Theorem 4.1, but the actual value of this quantity depends on the observed margin, hence the expression effective VC dimension.

Again, if we wish to take account of larger ranges and different values of extra technical analysis is necessary, but these details will not alter the overall shape of the result and will not be covered here. For more details see references in Section 4.8. We can view stratifying the result over different values of as assigning hypotheses to classes of different complexity depending on their margin. Hence, the classes are data dependent in contrast to classical structural risk minimisation when they must be specified before seeing the data. For this reason this type of result is sometimes referred to as data dependent structural risk minimisation.

We now turn our attention to the question of bounding the fat-shattering dimension for linear function classes, the final ingredient required if we wish to use the bounds for SVMs.

Theorem 4.16

Suppose that X is the ball of radius R in an inner product space $\mathbb{H}$, $X = \{\mathbf{x} \in \mathbb{H} : \|\mathbf{x}\|_{\mathbb{H}} \leq R\}$, and consider the class of functions

$$\mathscr{L} = \{\mathbf{x} \mapsto \langle \mathbf{w} \cdot \mathbf{x} \rangle : \|\mathbf{w}\|_{\mathbb{H}} \leq 1, \mathbf{x} \in X\}.$$

Then

$$\operatorname*{fat}_{\mathscr{L}}(\gamma) \leq \left(\frac{R}{\gamma}\right)^2.$$

The proof of this theorem follows from two intermediate results. The first states that if $S = \{\mathbf{x}_1, \ldots, \mathbf{x}\}$ is -shattered by , then every subset $S_0 \subset S$ satisfies

$$\left\| \sum S_0 - \sum (S - S_0) \right\|_{\mathbb{H}} \geq \ell\gamma, \qquad (4.5)$$

where the sum of a set means the sum of the vectors contained in that set. This result follows from considering the inner product of the vector inside the norm with the weight vector realising the classification determined by S_0 with margin . The second intermediate result computes the expected left hand side norm squared under a random choice of the subset S_0. If s is the $\{-1, 1\}$ vector indicating membership in S_0, we choose s uniformly at random and must estimate

$$\begin{aligned} E\left\| \sum S_0 - \sum (S - S_0) \right\|_{\mathbb{H}}^2 &= E\left\| \sum_{i=1}^{\ell} s_i \mathbf{x}_i \right\|_{\mathbb{H}}^2 \\ &= E\sum_{i=1}^{\ell} s_i^2 \|\mathbf{x}_i\|_{\mathbb{H}}^2 + 2E\sum_{i \neq j} s_i s_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle_{\mathbb{H}} \\ &= E\sum_{i=1}^{\ell} \|\mathbf{x}_i\|_{\mathbb{H}}^2 \leq R^2 \ell. \end{aligned}$$

Since there must be at least one S_0 with value less than or equal to the expectation, there exists at least one classification for which

$$\left\| \sum S_0 - \sum (S - S_0) \right\|_{\mathbb{H}} \leq R\sqrt{\ell}.$$

This inequality, together with inequality (4.5), shows that $R\sqrt{\ell} \geq \ell\gamma$, and the result follows.

Remark 4.17

Note that the bound on the fat-shattering dimension of linear learning machines is analogous to the mistake bound for the perceptron algorithm given in Theorem 2.3.

We are now in a position to quote an error bound for Support Vector Machines.

Theorem 4.18

Consider thresholding real-valued linear functions with unit weight vectors on an inner product space X and fix $\mathbb{R}^+$. For any probability distribution 퓓 on $X \times \{-1, 1\}$ with support in a ball of radius $\boldsymbol{R}$ around the origin, with probability 1 - over random examples S, any hypothesis f that has margin $m_S(f) \geq$ on S has error no more than

$$\operatorname*{err}_{\mathcal{D}}(f) \leq \varepsilon(\ell, \mathcal{L}, \delta, \gamma) = \frac{2}{\ell}\left(\frac{64R^2}{\gamma^2}\log\frac{e\ell\gamma}{4R}\log\frac{128\ell R^2}{\gamma^2} + \log\frac{4}{\delta}\right),$$

provided $> 2/$ and $64R^2/^2 <$.

The important qualitative aspect of this result is that the dimension of the input space does not appear, indeed the result also applies to infinite dimensional spaces. This type of result is sometimes said to be *dimension free*, as it suggests that the bound may overcome the curse of dimensionality. Based on the observations at the end of Section 4.2 we conclude that avoidance of the curse of dimensionality will only be possible if the distribution generating the examples is sufficiently benign and renders the task of identifying the particular target function correspondingly easier. In such cases the bound gives an assurance that with high probability we will make few errors on randomly chosen test examples. It is in this sense that we can view as providing a measure of how benign the distribution is and therefore how well we can expect to generalise. It is also possible to give a more refined estimate of the probability of misclassification in terms of the distance of the test point from the hyperplane - see Section 4.8 for pointers to references.

Theorem 4.18 becomes trivial and hence gives no information for the case where the data are non-separable or noise in the data causes the margin to be very small. The next two subsections discuss two methods which can handle these situations by taking a different measure of the margin distribution.

4.3.2 Margin Percentile Bounds

The next measure of the distribution of margin values that can be used to bound generalisation is a general percentile. This measure has the significant advantage that it includes the case when a hypothesis is not fully consistent with the training data. If we order the values in the margin distribution

$$M_S(f) = \{\gamma_i = y_i f(\mathbf{x}_i)\}$$

so that $_1 \leq _2 \leq \ldots \leq$ and fix a number $k <$, the $k/$ percentile $M_{S,k}(f)$ of $M_S(f)$ is $_k$. The following theorem provides a bound on the generalisation error in terms of $k/$ and $M_{S,k}(f)$.

Theorem 4.19

Consider thresholding real-valued linear functions with unit weight vectors on an inner-product space X and fix $\mathbb{R}^+$. There is a constant c, such that for any probability distribution 퓓 on $X \times \{-1, 1\}$ with support in a ball of radius R around the origin, with probability 1- over random examples S, any hypothesis f has error no more than

$$\operatorname*{err}_{\mathcal{D}}(f) \leq \frac{k}{\ell} + \sqrt{\frac{c}{\ell}\left(\frac{R^2}{M_{S,k}(f)^2}\log^2\ell + \log\frac{1}{\delta}\right)},$$

for all $k <$.

The proof of this theorem follows a similar pattern to that of Theorem 4.9 except that the counting of permutations of the double sample is complicated by the need to allow some mistakes on the left hand side.

The theorem suggests that we can obtain the best generalisation performance by minimising the number of margin errors, where we define a training point to be a *-margin error* if it has margin less than . The bound will be able to handle cases where there are a few outliers either causing the training set to be non-separable or forcing a very small margin. In these cases the margins of the difficult points can be ignored and the margin of the remaining points used. The cost incurred by moving to this larger margin is two-fold. Firstly the extra term *k/* takes into account the fact that we have ignored that fraction of the training set, while the second cost is the additional square root of the complexity term as compared with Theorem 4.18. The next subsection describes a bound in terms of the margin distribution that does not include the square root but rather depends on a measure of the size of the margin errors.

4.3.3 Soft Margin Bounds

We begin this subsection by giving a precise definition of the margin slack variables. This generalises Definition 2.6 to general function classes. For the case where is a class of linear functions the definition reduces back to Definition 2.6. The motivation is given by considering a target margin , and asking by how much individual points fail to meet this target. For points with margin larger than , this amount is zero, while for points that are misclassified the slack variable is greater than .

Definition 4.20

Consider using a class of real-valued functions on an input space *X* for classification by thresholding at 0. We define the *margin slack variable* of an example $(\mathbf{x}_i, y_i)$ $X \times \{-1, 1\}$ with respect to a function *f* and target margin to be the quantity (see Figure 2.4)

$$\xi((\mathbf{x}_i, y_i), f, \gamma) = \xi_i = \max(0, \gamma - y_i f(\mathbf{x}_i)).$$

Note that $_i >$ implies incorrect classification of $(\mathbf{x}_i, y_i)$. The *margin slack vector* (*S, f,*) of a training set

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$$

with respect to a function *f* and target margin contains the margin slack variables

$$\boldsymbol{\xi} = \boldsymbol{\xi}(S, f, \gamma) = (\xi_1, \ldots, \xi_\ell),$$

where the dependence on *S*, *f*, and is dropped when it is clear from the context.

We can think of the slack variables as measures of noise in the data which has caused individual training points to have smaller or even negative margins. The approach derived from taking account of the slack variables is suitable for handling noisy data.

We will derive a bound on the generalisation of the hypothesis *f* in terms of the target margin and different norms of the corresponding margin slack vector. The trick is to move the points that fail to meet the target margin outwards by embedding the input space into a larger space where a function can be found which increases their margins. The cost of performing the move can be measured by the norm of the margin slack vector. For an input space *X*, we use the auxiliary inner product space

$$L(X) = \left\{ f \in \mathbb{R}^X : \text{supp}(f) \text{ is countable and } \sum_{\mathbf{x} \in \text{supp}(f)} f(\mathbf{x})^2 < \infty \right\},$$

where for *f, g* *L(X)*, the inner product is given by

$$\langle f \cdot g \rangle = \sum_{\mathbf{x} \in \mathrm{supp}(f)} f(\mathbf{x}) g(\mathbf{x}).$$

We now embed the input space into the space $X \times L(X)$, using the mapping

$$\tau : \mathbf{x} \mapsto (\mathbf{x}, \delta_{\mathbf{x}})$$

where

$$\delta_{\mathbf{x}}(\mathbf{z}) = \begin{cases} 1, & \text{if } \mathbf{x} = \mathbf{z}, \\ 0, & \text{otherwise.} \end{cases}$$

Hence, a general function in g $L(X)$ that maps input $\mathbf{x}_i$ to a value c_i $\mathbb{R}$, $i = 1, 2, \ldots$, can be written

$$g = \sum_{i=1}^{\infty} c_i \delta_{\mathbf{x}_i}.$$

Since $L(X)$ is an inner product space we can view elements of $L(X)$ as linear functions on $L(X)$. Hence, for a function (f, g) $\times L(X)$, we define the action of (f,g) on $(\mathbf{x},\ )$ $X \times L(X)$ by

$$(f, g)(\mathbf{x}, \phi) = f(\mathbf{x}) + \langle g \cdot \phi \rangle,$$

so that the action of (f, g) on τ($\mathbf{x}$) is given by

$$(f, g)(\tau(\mathbf{x})) = f(\mathbf{x}) + \langle g \cdot \delta_{\mathbf{x}} \rangle.$$

Our strategy is to show that a judicious choice of g $L(X)$ causes the margin of the combined function to be , while the covering numbers of the expanded function class can be bounded in terms of the norm of the margin slack vector. We first show how to choose g so that the data are separated with margin . We define the following auxiliary function $g_f = g(S,f,\ )$ $L(X)$:

$$g_f = \sum_{i=1}^{\ell} \xi_i y_i \delta_{\mathbf{x}_i}.$$

Now for $(\mathbf{x}_i, y_i)$ S,

$$\begin{aligned} y_i(f, g_f)(\tau(\mathbf{x}_i)) &= y_i f(\mathbf{x}_i) + y_i \langle g_f \cdot \delta_{\mathbf{x}_i} \rangle \\ &= y_i f(\mathbf{x}_i) + y_i \sum_{j=1}^{\ell} \xi_j y_j \langle \delta_{\mathbf{x}_j} \cdot \delta_{\mathbf{x}_i} \rangle \\ &= y_i f(\mathbf{x}_i) + \xi_i y_i^2 \\ &= y_i f(\mathbf{x}_i) + \xi_i \geq \gamma. \end{aligned}$$

Hence, the augmented function does indeed have margin on the training set. For a point $\mathbf{x}$ not in the training set its action is identical to f,

$$\begin{aligned} (f, g_f)(\tau(\mathbf{x})) &= f(\mathbf{x}) + \sum_{j=1}^{\ell} \xi_j y_j \langle \delta_{\mathbf{x}_j} \cdot \delta_{\mathbf{x}} \rangle \\ &= f(\mathbf{x}), \end{aligned}$$

and so the generalisation performance of the two functions is identical. We therefore have the following result.

Theorem 4.21

Consider thresholding a real-valued function space and fix a subspace L $L(X)$, and $\mathbb{R}^+$. For any probability distribution 퓓 on $X \times \{-1, 1\}$, with probability 1 - over random examples S, any hypothesis f for which $g(S, f,)$ L has error no more than

$$\underset{\mathcal{D}}{\mathrm{err}}(f) \leq \varepsilon(\ell, \mathcal{F}, \delta, \gamma) = \frac{2}{\ell}\left(\log \mathcal{N}(\mathcal{F}, 2\ell, \gamma/4) + \log \mathcal{N}(L, 2\ell, \gamma/4) + \log \frac{2}{\delta}\right),$$

provided $> 2/$ and there is no discrete probability on misclassified training points.

Proof By the above we have that $(f, g(S, f,))$ has margin on the training set and equals f for points outside the training set. We can therefore apply Theorem 4.9 to bound the generalisation error for points not in the training set. It remains to place an appropriate bound on log 퓝(× L, 2 , /2). Let A be a cover of and B a cover of L each at scale /4 with respect to the 2 points $\mathbf{x}_1, \ldots, \mathbf{x}_2$. Then $A \times B$ is a /2 cover of × L with respect to the same points, since for general $(f, g$ × L,), we can find f A, such that

$$|f(\mathbf{x}_i) - f'(\mathbf{x}_i)| \leq \gamma/4, \; i = 1, \ldots, 2\ell,$$

and g B, such that

$$|g(\delta_{\mathbf{x}_i}) - g'(\delta_{\mathbf{x}_i})| \leq \gamma/4, \; i = 1, \ldots, 2\ell,$$

whence we have that

$$\begin{aligned} |(f, g)(\tau(\mathbf{x}_i)) - (f', g')(\tau(\mathbf{x}_i))| &\leq |f(\mathbf{x}_i) - f'(\mathbf{x}_i)| + |g(\delta_{\mathbf{x}_i}) - g'(\delta_{\mathbf{x}_i})| \\ &\leq \gamma/2, \; i = 1, \ldots, 2\ell. \end{aligned}$$

We conclude that

$$\mathcal{N}(\mathcal{F} \times L, 2\ell, \gamma/2) \leq \mathcal{N}(\mathcal{F}, 2\ell, \gamma/4)\mathcal{N}(L, 2\ell, \gamma/4),$$

and the result follows.

In order to apply this result we must choose appropriate sequences of sub-spaces of $L(X)$,

$$L_1 \subset L_2 \subset \ldots \subset L_k \subset \ldots \subset L(X),$$

and apply the theorem for each subspace, subsequently choosing the smallest L_k that contains $g(S, f,)$, for a given , S, and f. The two sequences that we will consider are those defined in terms of the 2-norm and 1-norm of the functions. In the case of the 2-norm, we have an inner produce space and can apply Theorem 4.16 and Lemma 4.13 to bound the covering numbers and obtain the following result.

Theorem 4.22

Consider thresholding real-valued linear functions with unit weight vectors on an inner product space X and fix $\mathbb{R}^+$. There is a constant c, such that for any probability distribution 퓓 on $X \times \{-1, 1\}$ with support in a ball of radius R around the origin, with probability 1 - over l random examples S, any hypothesis f has error no more than

$$\underset{\mathcal{D}}{\mathrm{err}}(f) \leq \frac{c}{\ell}\left(\frac{R^2 + \|\xi\|_2^2}{\gamma^2}\log^2 \ell + \log \frac{1}{\delta}\right),$$

where = $(f, S,)$ is the margin slack vector with respect to f and .

Remark 4.23

An analogous bound for the number of mistakes made in the first iteration of the perceptron algorithm is given in Theorem 2.7.

If the sequence L_k is defined in terms of the 1-norm then the bound obtained depends on this value together with an additional log factor.

Theorem 4.24

Consider thresholding real-valued linear functions with unit weight vectors on an inner product space X and fix $\mathbb{R}^+$. There is a constant c, such that for any probability distribution 퓓 on $X \times \{-1, 1\}$ with support in a ball of radius R around the origin, with probability 1- over random examples S, any hypothesis f has error no more than

$$\operatorname*{err}_{\mathcal{D}}(f) \leq \frac{c}{\ell}\left(\frac{R^2 + \|\xi\|_1^2 \log(1/\gamma)}{\gamma^2}\log^2 \ell + \log\frac{1}{\delta}\right),$$

where = (f, S,) is the margin slack vector with respect to f and .

The conclusion to be drawn from Theorems 4.22 and 4.24 is that the generalisation error bound takes into account the amount by which points fail to meet a target margin . The bound is in terms of a norm of the slack variable vector suggesting that this quantity should be minimised in order to optimise performance. The bound does not rely on the training points being linearly separable and hence can also handle the case when the data are corrupted by noise or the function class cannot capture the full complexity of the decision rule. Optimising the norm of the margin slack vector does not necessarily mean minimising the number of misclassifications. Hence, the inductive principle suggested by the theorems does not correspond to empirical risk minimisation. This fact will be important, as we shall see that minimising the number of misclassifications appears to be computationally more demanding than optimising the norms of the margin slack vector.

Optimising the norms of the margin slack vector has a diffuse effect on the margin. For this reason it is referred to as a *soft margin* in contrast to the maximal margin, which depends critically on a small subset of points and is therefore often called a *hard margin*. We will refer to the bound in terms of the 2-norm of the margin slack vector as the 2-norm soft margin bound, and similarly for the 1-norm soft margin.

10 Chapter 4: Generalisation Theory

4.4 Other Bounds on Generalisation and Luckiness

The previous section considered bounds on generalisation performance in terms of measures of the margin distribution. We argued that the bounds we must use in high dimensional spaces must be able to take advantage of favourable input distributions that are in some sense aligned with the target function. The bounds must avoid dependence on the dimension of the input space in favour of dependence on quantities measured as a result of the training algorithm, quantities that effectively assess how favourable the input distribution is. We described three results showing dependences on three different measures of the margin distribution. We will briefly argue in this section that bounds of this type do not necessarily have to depend on margin value. In particular the size of a sample compression scheme can be used to bound the generalisation by a relatively straightforward argument due to Littlestone and Warmuth. A sample compression scheme is defined by a fixed rule

$$\rho : S \mapsto \rho(S)$$

for constructing a classifier from a given set of labelled data. Given a large training set, it is compressed by finding the smallest subset (the compression set) $\hat{S} \subseteq S$ for which the reconstructed classifier $\rho(\hat{S})$ correctly classifies the whole set *S*. Fix a number *d* < . Suppose that for a particular training set we obtain a compressed set of size *d*. This can only occur in $\binom{\ell}{d}$ ways. For each such choice the probability that the resulting hypothesis has error more than , and yet correctly classifies the remaining - *d* randomly generated training points, is bounded by

$$(1-\varepsilon)^{\ell-d} \leq \exp\left(-\varepsilon(\ell-d)\right).$$

Hence, the probability that a compression set of size *d* has error greater than $_d$ can be bounded by

$$\binom{\ell}{d} \exp\left(-\varepsilon_d(\ell-d)\right). \qquad (4.6)$$

For

$$\varepsilon_d = \frac{1}{\ell-d}\left(d\ln\frac{e\ell}{d} + \ln\frac{\ell}{\delta}\right),$$

this will be less than / . It follows that the probability of $_d$ failing to bound the generalisation error for a compression set of size *d* is less than / and so the probability that the corresponding to the observed size of the compression set is greater than the generalisation error is at most . Hence, we have shown the following theorem.

Theorem 4.25

Consider a compression scheme *p*. For any probability distribution 퓓 on $X \times \{-1, 1\}$, with probability 1 - over random examples *S*, the hypothesis defined by a compression set of size *d* has error no more than

$$\operatorname*{err}_{\mathcal{D}}(f) \leq \frac{1}{\ell-d}\left(d\log\frac{e\ell}{d} + \log\frac{\ell}{\delta}\right).$$

We will see in Chapter 6 that the support vectors of a Support Vector Machine form a compression scheme that can reconstruct the maximal margin hyperplane. Theorem 4.25 therefore shows that the number of support vectors, a quantity that does not directly involve the margin, can also be used to measure how favourably the distribution generating the data is aligned with the target function, so giving another data dependent bound. This observation has led to the introduction of a general framework which uses a so-called *luckiness function* to assess how well the distribution is aligned with the target function. The size of the margin is just one such measure. Choosing a luckiness function corresponds to asserting a prior belief about

the type of relations that may occur between distributions and target functions. If that belief holds true we profit by a correspondingly improved generalisation, though of course there may be a small cost involved when the assumption fails.

4.5 Generalisation for Regression

The problem of regression is that of finding a function which approximates mapping from an input domain to the real numbers based on a training sample. The fact that the output values are no longer binary means that the mismatch between the hypothesis output and its training value will no longer be discrete. We refer to the difference between the two values as the *residual* of the output, an indication of the accuracy of the fit at this point. We must decide how to measure the importance of this accuracy, as small residuals may be inevitable while we wish to avoid large ones. The *loss function* determines this measure. Each choice of loss function will result in a different overall strategy for performing regression. For example least squares regression uses the sum of the squares of the residuals.

Although several different approaches are possible (see Section 4.8), we will provide an analysis for the generalisation performance of a regression function by using the bounds obtained for the classification case, as these will motivate the algorithms described in Chapter 6. Hence, we will consider a threshold test accuracy , beyond which we consider a mistake to have been made. We therefore aim to provide a bound on the probability that a randomly drawn test point will have accuracy less than . If we assess the training set performance using the same , we are effectively using the real-valued regressors as classifiers and the worst case lower bounds apply. What we must do in order to make use of the dimension free bounds is to allow a margin in the regression accuracy that corresponds to the margin of a classifier. We will use the same symbol to denote this margin, which measures the amount by which the training and test accuracy can differ. It should be emphasised that we are therefore using a different loss function during training and testing, where measures the discrepancy between the two losses, implying that a training point counts as a mistake if its accuracy is less than - . One way of visualising this method of assessing performance is to consider a band of size ±(-) around the hypothesis function. Any training points lying outside this band are considered to be training mistakes. Test points count as mistakes only if they lie outside the wider band of ± . Using this correspondence Theorem 4.18 has the following interpretation for regression estimation.

Theorem 4.26

Consider performing regression estimation with linear functions with unit weight vectors on an inner product space X and fix ≤ $\mathbb{R}^+$. For any probability distribution 퓓 on $X \times \mathbb{R}$ with support in a ball of radius R around the origin, with probability 1 - over random examples S, any hypothesis f , whose output is within - of the training value for all of the training set S, has residual greater than on a randomly drawn test point with probability at most

$$\operatorname*{err}_{\mathcal{D}}(f) \leq \varepsilon(\ell, \mathcal{L}, \delta, \gamma) = \frac{2}{\ell}\left(\frac{64R^2}{\gamma^2}\log\frac{e\ell\gamma}{4R}\log\frac{128\ell R^2}{\gamma^2} + \log\frac{4}{\delta}\right),$$

provided > 2/ and $64R^2/\ ^2 <$.

The theorem shows how we can bound the probability that the output of a unit norm linear function on a random test point will be out by more than provided its residuals on the training set are all smaller than - . Note that as in the classification case the dimension of the feature space does not enter into the formula, ensuring that the bound will be applicable even for the high dimensional feature spaces arising from the use of kernel functions.

We next consider the role that the margin slack variables play in the regression case.

Definition 4.27

Consider using a class of real-valued functions on an input space X for regression. We define the *margin slack variable* of an example $(\mathbf{x}_i, y_i)$ $X \times \mathbb{R}$ with respect to a function f , target accuracy and loss margin to be the quantity (see Figure 4.1 for a linear example and Figure 4.2 for a non-linear function)

$$\xi\left((\mathbf{x}_i, y_i), f, \theta, \gamma\right) = \xi_i = \max\left(0,\ |y_i - f(\mathbf{x}_i)| - (\theta - \gamma)\right).$$

Note that $_i >$ implies an error on $(\mathbf{x}_i, y_i)$ of more than . The *margin slack vector* $(S, f,\ ,\ )$ of a training set

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$$

with respect to a function *f* and target accuracy and loss margin contains the margin slack variables

$$\boldsymbol{\xi} = \boldsymbol{\xi}(S, f, \gamma) = (\xi_1, \ldots, \xi_\ell),$$

where the dependence on $S, f,\ ,$ and is dropped when it is clear from the context.

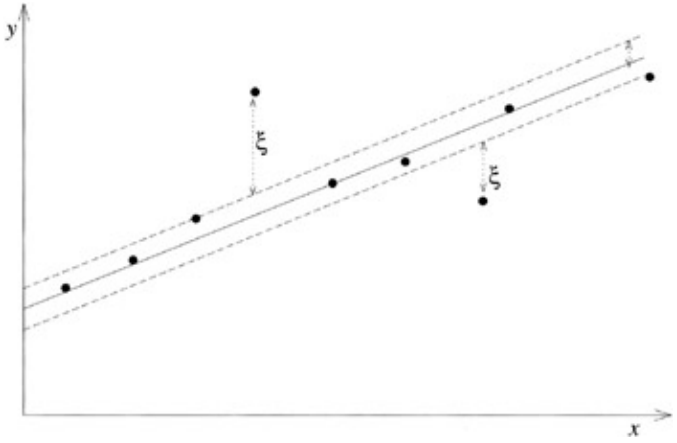


Figure 4.1: The slack variables for a one dimensional linear regression problem

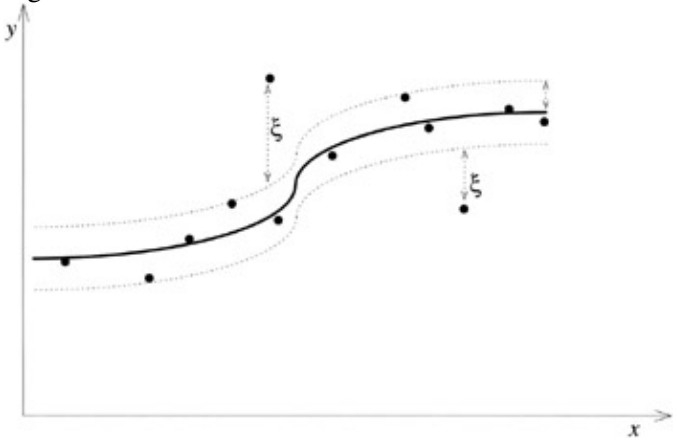


Figure 4.2: The slack variables for a non-linear regression function

Again the correspondence with the classification case is direct and leads to the following bounds on the generalisation performance of linear regressors in terms of the 2-norm of the slack variables. Note that in the case of regression it no longer makes sense to fix the norm of the weight vector, as in contrast to the classification case rescaling does result in a different functionality. The size of the norm of the weight vector affects the scale at which the covering must be sought for the equivalent unit weight vector of a linear function.

Theorem 4.28

Consider performing regression with linear functions on an inner product space *X* and fix ≤ $\mathbb{R}^+$. There is a constant *c*, such that for any probability distribution on $X \times \mathbb{R}$ with support in a ball of radius *R* around the origin, with probability 1 - over random examples *S*, the probability that a hypothesis **w** has output more than away from its true value is bounded by

$$\underset{\mathcal{D}}{\mathrm{err}}(f) \leq \frac{c}{\ell}\left(\frac{\|\mathbf{w}\|_2^2 R^2 + \|\boldsymbol{\xi}\|_2^2}{\gamma^2}\log^2 \ell + \log\frac{1}{\delta}\right),$$

where = (**w**, *S*, ,) is the margin slack vector with respect to **w**, , and .

The theorem is a significant advance over Theorem 4.26 since it can be applied to all linear functions and can take account of training points whose residuals lie outside the - tube. The 2-norm of these excess residuals enters into the formula together with the 2-norm of the linear function. If we consider the case when = , the 2-norm of the margin slack vector is the sum of the squares of the residuals on the training sequence, something often referred to as the *sum squared error (SSE)*. We therefore have the following corollary.

Corollary 4.29

Consider performing regression with linear functions on an inner product space *X* and fix $\mathbb{R}^+$. There is a constant *c*, such that for any probability distribution 퓓 on $X \times \mathbb{R}$ with support in a ball of radius *R* around the origin, with probability 1 - over random examples *S*, the probability that a hypothesis **w** has output more than away from its true value is bounded by

$$\underset{\mathcal{D}}{\mathrm{err}}(f) \leq \frac{c}{\ell}\left(\frac{\|\mathbf{w}\|_2^2 R^2 + SSE}{\theta^2}\log^2 \ell + \log\frac{1}{\delta}\right),$$

where SSE is the sum squared error of the function **w** on the training set *S*.

The corollary is directly applicable to least squares regression using linear functions, perhaps the most standard form of regression, but here used in a setting where the training sequence has been generated according to an unknown probability distribution. The resulting bound on the probability that a test point has residual greater than is a novel way of assessing performance of such functions. In Chapter 6 we will see that the ridge regression algorithm discussed in Subsection 2.2.2 directly optimises this bound and hence potentially outperforms the standard least squares algorithm.

Finally, we can translate the 1-norm bound of Theorem 4.24 to obtain the following result.

Theorem 4.30

Consider performing regression with linear functions on an inner product space *X* and fix ≤ $\mathbb{R}^+$. There is a constant *c*, such that for any probability distribution 퓓 on $X \times \mathbb{R}$ with support in a ball of radius *R* around the origin, with probability 1 - over random examples *S*, the probability that a hypothesis **w** has output more than away from its true value is bounded by

$$\underset{\mathcal{D}}{\mathrm{err}}(f) \leq \frac{c}{\ell}\left(\frac{\|\mathbf{w}\|_2^2 R^2 + \|\boldsymbol{\xi}\|_1^2 \log(1/\gamma)}{\gamma^2}\log^2 \ell + \log\frac{1}{\delta}\right),$$

where = (**w**,*S*, ,) is the margin slack vector with respect to **w**, , and .

The bound in terms of the 1-norm of the slacks and 2-norm of the weight vector may seem an unnatural mix of two different norms. However, the use of 2-norm for the linear function is dictated by our prior over the function class, while the norm of the slack variables should be chosen to model the type of noise that has corrupted the training examples. For example, if we optimise the 1-norm bound the resulting regressor takes less account of points that have large residuals and so can handle outliers better than by optimising the 2-norm bound.

---

4 Chapter 4: Generalisation Theory

4.6 Bayesian Analysis of Learning

In this section we will briefly review the Bayesian approach to learning. Its motivation does not come from bounds on the generalisation and so the section may seem out of place in this chapter. Though Bayesian analysis can be used to estimate generalisation, in this section we only cover that part of the theory needed to motivate the learning strategy. The pac style of analysis we have considered in the earlier sections of this chapter has focused on finding an error bound that will hold with high probability. That approach can be seen as conservative in that it attempts to find a bound on the error probability that will hold with high confidence. The Bayesian approach in contrast attempts to choose output values that are most likely based on the observed training values. This can result in a function not actually in the initial set of hypotheses. Alternatively if we restrict ourselves to choosing a function from the set, it can motivate an optimal choice. It therefore is attempting to make the best possible choice based on the available data. In order to make such a desirable calculation tractable a number of assumptions need to be made including the existence of a prior distribution over the set of hypotheses and a (Gaussian) noise model. These assumptions can render the choice of function less reliable when compared with the pac analysis, which depends only on the assumed existence of an underlying probability distribution that generates the training and test data independently. Nonetheless we shall see that despite its very different starting point the resulting computation and function are very closely related to the support vector approach.

As described in Chapter 3 the prior distribution can be described by a Gaussian process and choosing its covariance function defines the prior in a similar way to that in which choosing the kernel for a Support Vector Machine defines the feature space. The aim of Bayesian analysis is to update that distribution based on the observation of the particular data. The more a particular data point disagrees with a function the more the posterior probability of that function is reduced. Bayes' formula,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)},$$

is used to calculate this altered probability under the assumption that the errors between the true function and the observed outputs are generated by a Gaussian distribution with mean 0 and variance ^{2}. Once the updated or a posteriori distribution based on all the observations has been calculated, the learner can either pick the function with highest likelihood or take an average of the outputs over the different functions weighted according to the posterior distribution. The function with highest likelihood is often referred to as the maximum a posteriori or *MAP* hypothesis. The second more powerful approach corresponds to choosing the most likely output given a test input but with no restrictions placed on what function is used. In fact the function will always be in the convex closure of the original set of hypotheses, since it is an average of those functions weighted according the a posteriori distribution. In the case of Gaussian processes the two approaches coincide since a linear class is equal to its own convex closure and the a posteriori distribution is Gaussian.

We restrict consideration to regression since the assumption of a Gaussian distribution of errors is only meaningful in this case. In order to be consistent with our previous notation we will use y_i, $i = 1,\ldots,$, to denote the output values from the training set that are assumed to be corrupted by noise. The underlying true target output values will be denoted by t_i, $i = 1,\ldots,$. Unfortunately, this notation is the reverse of that adopted in some of the literature. The relation between the vectors **y** and **t** is assumed to be Gaussian,

$$P(\mathbf{y}|\mathbf{t}) \propto \exp\left[-\frac{1}{2}(\mathbf{y}-\mathbf{t})'\Omega^{-1}(\mathbf{y}-\mathbf{t})\right],$$

where Ω = $^2\mathbf{I}$. As mentioned above, the aim of the Bayesian analysis is to calculate the probability distribution for the true output t given a novel input **x**, and a training set S, which we split into the matrix **X** of input vectors and vector **y** of corresponding outputs. Hence, we wish to estimate $P(t|\mathbf{x}, S)$ and in particular find where it achieves its maximum. First we use Bayes' formula to express

$$P(t, \mathbf{t}|\mathbf{x}, S) = P(t, \mathbf{t}|\mathbf{y}, \mathbf{x}, \mathbf{X}) = \frac{P(\mathbf{y}|\mathbf{t}, \mathbf{x}, \mathbf{X})P(t, \mathbf{t}|\mathbf{x}, \mathbf{X})}{P(\mathbf{y}|\mathbf{x}, \mathbf{X})}$$
$$= \frac{P(\mathbf{y}|\mathbf{t})P(t, \mathbf{t}|\mathbf{x}, \mathbf{X})}{P(\mathbf{y}|\mathbf{x}, \mathbf{X})} \propto P(\mathbf{y}|\mathbf{t})P(t, \mathbf{t}|\mathbf{x}, \mathbf{X}),$$

where we have treated the denominator as a constant since it does not depend on our choice of hypothesis and hence does not vary once the training set and test point are known. The second factor in the final expression is the prior distribution for the true outputs given a set of inputs with no knowledge about the output values contained in the training set. This will be determined by the choice of Gaussian process through a covariance function. The first factor is the weighting given to a particular hypothesis identified by its output values on the training set inputs. The weighting is based entirely on the discrepancy between the hypothesis outputs and the values given in the training set. The task remaining for the Bayesian learner is to 'marginalise' over $\mathbf{t}$, by which is meant summing the probability of a particular value for t over all possible values that the parameters $\mathbf{t}$ might take. The advantage of using Gaussian distributions is that the resulting distribution is also Gaussian and that its mean and variance can be calculated analytically, hence giving the maximum value for t together with what can be regarded as an error bar on the accuracy of the prediction. Chapter 6 will describe this calculation and compare the resulting decision rule with that obtained using Support Vector Machines.

4.7 Exercises

1. Prove Proposition 4.5.
2. Describe how Theorem 4.9 could be made to apply for all values of , indicating what weakening of the bound would result.
3. In Section 4.4 consider choosing $_d$ so that formula (4.6) is less than $_d$ for some values $_d$ satisfying

$$\sum_{i=1}^{\ell} \delta_i = 1.$$

Show that a generalisation of Theorem 4.25 results. Given that we know that the probability of obtaining d support vectors is p_d, what choice of $_d$ will give the best expected bound in the generalised theorem?

4.8 Further Reading and Advanced Topics

The analysis of generalisation based on the VC dimension was developed by Vapnik and Chervonenkis starting from the mid 1960s [162]. It has been successfully applied in different fields, motivating for example the development in statistics of a uniform law of large numbers [117]. Most of its basic theoretical results were already present in Vapnik's book [157]. The development of the pac model of machine learning, on the other hand, can be traced to the seminal paper by Valiant in 1984 [155], which laid the foundations of what became known as computational learning theory: a large set of models describing, among others, on-line learning, query learning, unsupervised and supervised learning, and recently also applied to reinforcement learning. The introduction of the VC results within this theory, together with the lower bounds, is contained in the landmark paper by Blumer et al. [18], and has greatly influenced the entire field of machine learning. VC theory has since been used to analyse the performance of learning systems as diverse as decision trees, neural networks, and others; many learning heuristics and principles used in practical applications of machine learning have been explained in terms of VC theory.

Many introductory books to computational learning theory exist, for example [6], [71], although they are often mainly focused on the pac model, somewhat neglecting the rich fields of on-line, query, and unsupervised learning. A good introduction to the statistical analysis of pattern recognition is given by Devroye et al. [33]. VC theory has recently also come to be known as statistical learning theory, and is extensively described in the recent book of Vapnik [159], and in other books that preceded it [157] [158], as well as earlier papers by Vapnik and Chervonenkis [162][164][165]. An easy introduction to the theory is provided by [169], and a complete account, including also very recent results, can be found in [5]. An international conference is held every year on computational learning theory, known as the COLT conference, where new results are presented. Websites such as http://www.neurocolt.org offer large repositories of recent papers.

The question of how the margin might affect generalisation was raised by many researchers including Duda and Hart [35], Vapnik and Chervonenkis [166], and Mangasarian. Vapnik and Chervonenkis [163] obtained bounds for the case when the margin is measured on the combined training and test sets using a quantity analogous to the fat-shattering dimension. The fat-shattering dimension itself (sometimes also called the scale-sensitive dimension, or V dimension) has appeared implicitly in a number of earlier references but was introduced into computational learning theory by [72] and shown by Alon et al. [2] to characterise the so-called Glivenko Cantelli classes. The fat-shattering dimension of linear classifiers was obtained by different authors ([54] and [138]), while the proof presented in this chapter appears in [9].

The first papers to prove the large margin results were [138], [10] with the second reference containing the percentile result of Subsection 4.3.2. The soft margin bounds involving the margin slack vector for both classification and regression are due to [141], [142], [139], [140] and use techniques similar to those discussed in Chapter 2 for obtaining mistake bounds in the non-separable case (see Section 2.5 for further references). Those results are summarised in [9] and [149]. A quantity related to the margin slack vector is the so-called ‘hinge loss', used to obtain mistake bounds in the on-line learning framework [48]. Margin analysis has since been applied to describe systems like Adaboost [127], Bayesian classifiers [32], decision trees [12], neural networks [10], and is now a standard tool in machine learning. The margin analysis has also been extended to take account of the margin of the test example [137].

Anthony and Bartlett used the fat-shattering dimension to obtain results for regression similar Theorem 4.26. Different analyses of generalisation are possible for regression, such as in [159]. The book [5] provides an excellent introduction to the analysis of regression.

The reason why margin analysis requires different tools from VC theory is that the quantity used to characterise the richness of a hypothesis class, the margin, depends on the data. Only after training the learning machine can one know what is the complexity of the resulting hypothesis. This style of analysis, which provides a way of exploiting benign collusions between the target function and input distribution, is often called data dependent analysis, or data dependent structural risk minimisation. The first data dependent

result was Theorem 4.25 on the generalisation power of compression schemes and which is due to Littlestone and Warmuth [79][42], while the paper [138] introduced the general luckiness framework mentioned in Section 4.4. Other data dependent results include micro-choice algorithms, and pac-Bayesian bounds [93][94]. More recent bounds include [133] and [37], who like [138] bring out the connection between classification and regression.

Bayesian analysis is a traditional field within statistics, and has been applied to pattern recognition for a long time [35]. In recent years, a new surge of interest in Bayesian analysis has come from the neural networks community, mainly thanks to the work of MacKay [82]. An introduction to such analysis is provided by the books of Bishop [16] and Neal [102]. More recently, attention has been directed at Gaussian processes, a standard tool in statistics, described in [120] and [180]. We will return to the subject of Gaussian processes in Chapter 6. A Bayesian analysis of generalisation of Gaussian processes has been performed by Sollich [150] and Opper and Vivarelli [106]. Other analyses of generalisation are possible, based on statistical mechanics (see for example [105]), or on the theory of on-line algorithms [75].

These references are also given on the website http://www.support-vector.net, which will be kept up to date with new work, pointers to software and papers that are available on-line.

Chapter 5: Optimisation Theory

Overview

All of the inductive strategies presented in Chapter 4 have a similar form. The hypothesis function should be chosen to minimise (or maximise) a certain functional. In the case of linear learning machines (LLMs), this amounts to finding a vector of parameters that minimises (or maximises) a certain cost function, typically subject to some constraints. Optimisation theory is the branch of mathematics concerned with characterising the solutions of classes of such problems, and developing effective algorithms for finding them. The machine learning problem has therefore been converted into a form that can be analysed within the framework of optimisation theory.

Depending on the specific cost function and on the nature of the constraints, we can distinguish a number of classes of optimisation problems that are well understood and for which efficient solution strategies exist. In this chapter we will describe some of the results that apply to cases in which the cost function is a convex quadratic function, while the constraints are linear. This class of optimisation problems are called convex quadratic programmes, and it is this class that proves adequate for the task of training SVMs.

Optimisation theory will not only provide us with algorithmic techniques, but also define the necessary and sufficient conditions for a given function to be a solution. An example of this is provided by the theory of duality, which will provide us with a natural interpretation of the dual representation of LLMs presented in the previous chapters. Furthermore, a deeper understanding of the mathematical structure of solutions will inspire many specific algorithmic heuristics and implementation techniques described in Chapter 7.

5.1 Problem Formulation

The general form of the problem considered in this chapter is that of finding the maximum or minimum of a function, typically subject to some constraints. The general optimisation problem can be stated as follows:

Definition 5.1

(Primal optimisation problem) Given functions f, g_i, $i = 1,\ldots,k$, and h_i, $i = 1,\ldots,m$, defined on a domain Ω $\mathbb{R}^n$,

$$\begin{array}{lll} \text{minimise} & f(\mathbf{w}), & \mathbf{w} \in \Omega, \\ \text{subject to} & g_i(\mathbf{w}) \leq 0, & i = 1,\ldots,k, \\ & h_i(\mathbf{w}) = 0, & i = 1,\ldots,m, \end{array}$$

where $f(\mathbf{w})$ is called the *objective function*, and the remaining relations are called, respectively, the *inequality* and *equality constraints*. The optimal value of the objective function is called the *value of the optimisation problem*.

To simplify the notation we will write $\mathbf{g}(\mathbf{w}) \leq \mathbf{0}$ to indicate $g_i(\mathbf{w}) \leq 0$, $i = 1,\ldots,k$. The expression $\mathbf{h}(\mathbf{w}) = \mathbf{0}$ has a similar meaning for the equality constraints.

Since maximisation problems can be converted to minimisation ones by reversing the sign of $f(\mathbf{w})$, the choice of minimisation does not represent a restriction. Similarly any constraints can be rewritten in the above form.

The region of the domain where the objective function is defined and where all the constraints are satisfied is called the *feasible region*, and we will denote it by

$$R = \{\mathbf{w} \in \Omega : \mathbf{g}(\mathbf{w}) \leq \mathbf{0}, \mathbf{h}(\mathbf{w}) = \mathbf{0}\}.$$

A solution of the optimisation problem is a point $\mathbf{w}^*$ R such that there exists no other point $\mathbf{w}$ R for which $f(\mathbf{w}) < f(\mathbf{w}^*)$. Such a point is also known as a global minimum. A point $\mathbf{w}^*$ Ω is called a *local minimum* of $f(\mathbf{w})$ if ∃ > 0 such that $f(\mathbf{w}) \geq f(\mathbf{w}^*)$, ∀$\mathbf{w}$ Ω such that ⊻⊻$\mathbf{w} - \mathbf{w}^*$⊻⊻ < .

Different assumptions on the nature of the objective function and the constraints create different optimisation problems.

Definition 5.2

An optimisation problem in which the objective function, inequality and equality constraints are all linear functions is called a *linear programme*. If the objective function is quadratic while the constraints are all linear, the optimisation problem is called a *quadratic programme*.

An inequality constraint $g_i(\mathbf{w}) \leq 0$ is said to be *active* (or tight) if the solution $\mathbf{w}^*$ satisfies $g_i(\mathbf{w}^*) = 0$, otherwise it is said to be *inactive*. In this sense, equality constraints are always active. Sometimes, quantities called *slack variables* and denoted by are introduced, to transform an inequality constraint into an equality one, as follows:

$$g_i(\mathbf{w}) \leq 0 \iff g_i(\mathbf{w}) + \xi_i = 0, \text{ with } \xi_i \geq 0.$$

Slack variables associated with active constraints are equal to zero, while those for inactive constraints indicate the amount of 'looseness' in the constraint.

We will consider restricted classes of optimisation problems. First we define what are meant by a convex

function and a convex set.

Definition 5.3

A real-valued function $f(\mathbf{w})$ is called *convex* for $\mathbf{w} \quad \mathbb{R}^n$ if, ∀**w**, **u** $\mathbb{R}^n$, and for any (0,1),

$$f(\theta\mathbf{w} + (1-\theta)\mathbf{u}) \leq \theta f(\mathbf{w}) + (1-\theta)f(\mathbf{u}).$$

If a strict inequality holds, the function is said to be *strictly convex*. A function that is twice differentiable will be convex provided its Hessian matrix is positive semi-definite. An *affine* function is one that can be expressed in the form

$$f(\mathbf{w}) = \mathbf{A}\mathbf{w} + \mathbf{b},$$

for some matrix **A** and vector **b**. Note that affine functions are convex as they have zero Hessian. A set Ω $\mathbb{R}^n$ is called *convex* if, ∀**w, u** Ω, and for any (0, 1), the point (**w** + (1 −)**u**) Ω.

If a function f is convex, any local minimum $\mathbf{w}^*$ of the unconstrained optimisation problem with objective function f is also a global minimum, since for any $\mathbf{u} \neq \mathbf{w}^*$, by the definition of a local minimum there exists sufficiently close to 1 that

$$\begin{aligned} f(\mathbf{w}^*) &\leq f(\theta\mathbf{w}^* + (1-\theta)\mathbf{u}) \\ &\leq \theta f(\mathbf{w}^*) + (1-\theta)f(\mathbf{u}). \end{aligned}$$

It follows that $f(\mathbf{w}^*) < f(\mathbf{u})$. It is this property of convex functions that renders optimisation problems tractable when the functions and sets involved are convex.

Definition 5.4

An optimisation problem in which the set Ω, the objective function and all of the constraints are convex is said to be *convex*.

For the purposes of training SVMs we can restrict ourselves to the case where the constraints are linear, the objective function is convex and quadratic and Ω = $\mathbb{R}^n$, hence we consider convex quadratic programmes.

Optimisation theory is concerned both with describing basic properties that characterise the optimal points, and with the design of algorithms for obtaining solutions. In this chapter we will focus on the theoretical aspects, leaving algorithmic considerations to be addressed in Chapter 7. The next section will present the technique of Lagrange multipliers and its extensions, always restricted to the case of convex quadratic programmes.

5.2 Lagrangian Theory

The purpose of Lagrangian theory is to characterise the solution of an optimisation problem initially when there are no inequality constraints. The main concepts of this theory are the Lagrange multipliers and the Lagrangian function. This method was developed by Lagrange in 1797 for mechanical problems, generalising a result of Fermat from 1629. In 1951 Kuhn and Tucker further extended the method to allow inequality constraints in what is known as Kuhn–Tucker theory. These three increasingly general results will provide all that we need to develop efficient solutions for the task of optimising SVMs. For ease of understanding we first introduce the simplest case and then go on to consider the more complex type of problems. When there are no constraints the stationarity of the objective function is sufficient to characterise the solution.

Theorem 5.5

(Fermat) A necessary condition for $\mathbf{w}^*$ to be a minimum of $f(\mathbf{w})$, $f \quad C^1$, is $\frac{\partial f(\mathbf{w}^*)}{\partial \mathbf{w}} = \mathbf{0}$. This condition, together with convexity of f, is also a sufficient condition.

We will give one simple example of this type of optimisation taken from Chapter 3 when we considered finding the best approximation in a reproducing kernel Hilbert space.

Example 5.6

Suppose we wish to perform regression from a training set

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)) \subset (X \times Y)^\ell \subset (\mathbb{R}^n \times \mathbb{R})^\ell,$$

generated from the target function $t(\mathbf{x})$. If we assume a dual representation of the form

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}),$$

in Example 3.11 we showed that to minimise the RKHS norm of the error we must minimise

$$-2\sum_{i=1}^{\ell} \alpha_i y_i + \sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j).$$

The positive semi-definiteness of the kernel K ensures that the objective function is convex. Using Theorem 5.5 we compute the derivatives with respect to $_i$ and set equal to zero, obtaining

$$-2y_i + 2\sum_{j=1}^{\ell} \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) = 0, \; i = 1, \ldots, \ell,$$

or

$$\mathbf{G}\boldsymbol{\alpha} = \mathbf{y},$$

where we have denoted by $\mathbf{G}$ the Gram matrix with entries $\mathbf{G}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$. Hence, the parameters * for the solution can be obtained as

$$\boldsymbol{\alpha} = \mathbf{G}^{-1}\mathbf{y}.$$

In constrained problems, one needs to define a function, known as the Lagrangian, that incorporates information about both the objective function and the constraints, and whose stationarity can be used to detect

solutions. Precisely, the Lagrangian is defined as the objective function plus a linear combination of the constraints, where the coefficients of the combination are called the Lagrange multipliers.

Definition 5.7

Given an optimisation problem with objective function $f(\mathbf{w})$, and equality constraints $h_i(\mathbf{w}) = 0$, $i = 1,\ldots,m$, we define the *Lagrangian function* as

$$L(\mathbf{w}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w})$$

where the coefficients β$_i$ are called the *Lagrange multipliers*.

If a point **w*** is a local minimum for an optimisation problem with only equality constraints, it is possible that $\frac{\partial f(\mathbf{w}^*)}{\partial \mathbf{w}} \neq \mathbf{0}$, but that the directions in which we could move to reduce f cause us to violate one or more of the constraints. In order to respect the equality constraint h, we must move perpendicular to $\frac{\partial h_i(\mathbf{w}^*)}{\partial \mathbf{w}}$, and so to respect all of the constraints we must move perpendicular to the subspace V spanned by

$$\left\{ \frac{\partial h_i(\mathbf{w}^*)}{\partial \mathbf{w}} : i = 1, \ldots, m \right\}.$$

If the $\frac{\partial h_i(\mathbf{w}^*)}{\partial \mathbf{w}}$ are linearly independent no legal move can change the value of the objective function, whenever $\frac{\partial f(\mathbf{w}^*)}{\partial \mathbf{w}}$ lies in the subspace V or in other words when there exist β such that
i

$$\frac{\partial f(\mathbf{w}^*)}{\partial \mathbf{w}} + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w}^*) = 0.$$

This observation forms the basis of the second optimisation result concerning optimisation problems with equality constraints.

Theorem 5.8

(Lagrange) A necessary condition for a normal point **w*** to be a minimum of $f(\mathbf{w})$ subject to $h_i(\mathbf{w}) = 0$, $i = 1,\ldots,m$, with $f,\ h_i \quad C^1$, is

$$\begin{aligned} \frac{\partial L(\mathbf{w}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{w}} &= \mathbf{0}, \\ \frac{\partial L(\mathbf{w}^*, \boldsymbol{\beta}^*)}{\partial \beta} &= 0, \end{aligned}$$

for some values **β***. The above conditions are also sufficient provided that $L(\mathbf{w}, \boldsymbol{\beta}^*)$ is a convex function of **w**.

The first of the two conditions gives a new system of equations, whereas the second returns the equality constraints. By imposing the conditions (jointly solving the two systems) one obtains the solution.

Example 5.9

(Largest volume box with a given surface) Let us consider the problem of finding the dimensions of the sides of a box, w, u, v whose volume is maximal and whose surface is equal to a given value c. The problem can be rewritten as

$$\begin{aligned} &\text{minimise} && -wuv \\ &\text{subject to} && wu + uv + vw = c/2. \end{aligned}$$

The Lagrangian of this problem is $L = -wuv + \β(wu + uv + vw - c/2)$ and the necessary conditions for optimality are given by the constraints and by the stationarity conditions

$$\begin{aligned} \frac{\partial L}{\partial w} &= -uv + \beta(u+v) = 0, \\ \frac{\partial L}{\partial u} &= -vw + \beta(v+w) = 0, \\ \frac{\partial L}{\partial v} &= -wu + \beta(w+u) = 0. \end{aligned}$$

These conditions imply that $\βv(w-u) = 0$ and $\βw(u-v) = 0$, whose only non-trivial solution is $w = u = v = \sqrt{\frac{c}{6}}$. Hence since the conditions are necessary for a minimum and the trivial solutions have zero volume, the maximum volume box is a cube.

Example 5.10

(Maximum entropy distribution) The entropy of a probability distribution $\mathbf{p} = (p_1,\ldots,p_n)$ over a finite set $\{1,2,\ldots,n\}$ is defined as $H(\mathbf{p}) = -\sum_{i=1}^{n} p_i \log p_i$, where naturally $\sum_{i=1}^{n} p_i = 1$. The distribution with maximum entropy can be found by solving an optimisation problem with the Lagrangian

$$L(\mathbf{p}, \beta) = \sum_{i=1}^{n} p_i \log p_i + \beta \left(\sum_{i=1}^{n} p_i - 1 \right)$$

over the domain $\Ω = \{\mathbf{p} : p_i \geq 0,\ i = 1,\ldots,n\}$. The stationarity conditions imply that $\log ep_i + \β = 0$ for all i, indicating that all p need to be equal to $\frac{2^{-\beta}}{e}$. This together with the constraint gives $\mathbf{p} = (\frac{1}{n}, \ldots, \frac{1}{n})$. Since $\Ω$ is convex, the constraint is affine and the objective function is convex, having a diagonal Hessian with entries $(ep_i \ln 2)^{-1}$, this shows that the uniform distribution has the maximum entropy.

Remark 5.11

Note that if we replace the ith constraint by $h_i(\mathbf{w}) = b_i$, and consider the value of the objective function at the optimal solution $f^* = f(\mathbf{w}^*)$ as a function of b, then $\left[\frac{\partial f^*}{\partial b_i}\right]_{b_i=0} = \beta_i^*$. Hence the Lagrange multipliers contain information about the sensitivity of the solution to a given constraint.

Remark 5.12

Note that since the constraints are equal to zero, the value of the Lagrangian at the optimal point is equal to the value of the objective function

$$L(\mathbf{w}^*, \boldsymbol{\beta}^*) = f(\mathbf{w}^*).$$

We now consider the most general case where the optimisation problem contains both equality and inequality constraints. We first give the definition of the generalised Lagrangian.

Definition 5.13

Given an optimisation problem with domain $\Ω \quad \mathbb{R}^n$,

$$\begin{array}{lll} \text{minimise} & f(\mathbf{w}), & \mathbf{w} \in \Omega, \\ \text{subject to} & g_i(\mathbf{w}) \leq 0, & i = 1, \ldots, k, \\ & h_i(\mathbf{w}) = 0, & i = 1, \ldots, m, \end{array}$$

we define the *generalised Lagrangian function* as

$$\begin{aligned} L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w}) \\ &= f(\mathbf{w}) + \boldsymbol{\alpha}' \mathbf{g}(\mathbf{w}) + \boldsymbol{\beta}' \mathbf{h}(\mathbf{w}). \end{aligned}$$

We can now define the Lagrangian dual problem.

Definition 5.14

The *Lagrangian dual problem* of the primal problem of Definition 5.1 is the following problem:

$$\begin{array}{ll} \text{maximise} & \theta(\boldsymbol{\alpha}, \boldsymbol{\beta}), \\ \text{subject to} & \boldsymbol{\alpha} \geq \mathbf{0}, \end{array}$$

where (, β) = inf$_{w}$ Ω L(**w**, , β). The value of the objective function at the optimal solution is called the *value of the problem*.

We begin by proving a theorem known as the weak duality theorem, which gives one of the fundamental relationships between the primal and dual problems and has two useful corollaries.

Theorem 5.15

Let w Ω be a feasible solution of the primal problem of Definition 5.1 and (, β) a feasible solution of the dual problem of Definition 5.14. Then f(**w**) ≥ (,β).

Proof From the definition of (, β) for **w** Ω we have

$$\begin{aligned} \theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) &= \inf_{\mathbf{u} \in \Omega} L(\mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\ &\leq L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\ &= f(\mathbf{w}) + \boldsymbol{\alpha}' \mathbf{g}(\mathbf{w}) + \boldsymbol{\beta}' \mathbf{h}(\mathbf{w}) \leq f(\mathbf{w}), \qquad (5.1) \end{aligned}$$

since the feasibility of **w** implies **g**(**w**) ≤ **0** and **h**(**w**) = **0**, while the feasibility of (, β) implies ≥ **0**.

Corollary 5.16

The value of the dual is upper bounded by the value of the primal,

$$\sup \{\theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) : \boldsymbol{\alpha} \geq \mathbf{0}\} \leq \inf \{f(\mathbf{w}) : \mathbf{g}(\mathbf{w}) \leq \mathbf{0}, \mathbf{h}(\mathbf{w}) = \mathbf{0}\}.$$

Corollary 5.17

If f(**w***) = (*, β*), where * ≥ **0**, and **g**(**w***) ≤ **0**, **h**(**w***) = **0**, then **w*** and (*, β*) solve the primal and dual problems respectively. In this case $_i g_i$(**w***) = 0, for i = 1,…,k.

Proof Since the values are equal the sequence of inequalities in equation (5.1) must become equalities. In particular the last inequality can only be an equality if *$_i g_i$(**w***) = 0, for all i.

Remark 5.18

Hence, if we attempt to solve the primal and dual problems in tandem, we may be able to detect that we have reached the solution by comparing the difference between the values of the primal and dual problems. If this reduces to zero, we have reached the optimum. This approach relies on the solutions of the primal and dual having the same value, something that is not in general guaranteed. The difference between the values of the primal and dual problems is known as the *duality gap*.

Another way of detecting the absence of a duality gap is the presence of a *saddle point*. A saddle point of the Lagrangian function for the primal problem is a triple

$$(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*), \text{ with } \mathbf{w}^* \in \Omega, \boldsymbol{\alpha}^* \geq \mathbf{0},$$

satisfying the additional property that

$$L(\mathbf{w}^*, \boldsymbol{\alpha}, \boldsymbol{\beta}) \leq L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) \leq L(\mathbf{w}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*),$$

for all **w** Ω, ≥ **0**. Note that **w** here is not required to satisfy the equality or inequality constraints.

Theorem 5.19

The triple (**w***, *, β*) is a saddle point of the Lagrangian function for the primal problem, if and only if its components are optimal solutions of the primal and dual problems and there is no duality gap, the primal and dual problems having the value

$$f(\mathbf{w}^*) = \theta(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*).$$

We will now quote the strong duality theorem, which guarantees that the dual and primal problems have the same value for the optimisation problems that we will be considering.

Theorem 5.20

(Strong duality theorem) Given an optimisation problem with convex domain Ω $\mathbb{R}^n$,

$$\begin{array}{lll} \textit{minimise} & f(\mathbf{w}), & \mathbf{w} \in \Omega, \\ \textit{subject to} & g_i(\mathbf{w}) \leq 0, & i = 1, \ldots, k, \\ & h_i(\mathbf{w}) = 0, & i = 1, \ldots, m, \end{array}$$

where the g_i and h_i are affine functions, that is

$$\mathbf{h}(\mathbf{w}) = \mathbf{A}\mathbf{w} - \mathbf{b},$$

for some matrix **A** and vector **b**, the duality gap is zero.

We are now in a position to give the Kuhn–Tucker theorem giving conditions for an optimum solution to a general optimisation problem.

Theorem 5.21

(Kuhn–Tucker) Given an optimisation problem with convex domain Ω $\mathbb{R}^n$,

$$\begin{array}{lll} \textit{minimise} & f(\mathbf{w}), & \mathbf{w} \in \Omega, \\ \textit{subject to} & g_i(\mathbf{w}) \leq 0, & i = 1, \ldots, k, \\ & h_i(\mathbf{w}) = 0, & i = 1, \ldots, m, \end{array}$$

with f C^1 convex and g_i, h_i affine, necessary and sufficient conditions for a normal point **w*** to be an optimum are the existence of ***, β*** such that

$$\frac{\partial L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{w}} = \mathbf{0},$$
$$\frac{\partial L(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \beta} = 0,$$
$$\alpha_i^* g_i(\mathbf{w}^*) = 0, \; i = 1, \ldots, k,$$
$$g_i(\mathbf{w}^*) \leq 0, \; i = 1, \ldots, k,$$
$$\alpha_i^* \geq 0, \; i = 1, \ldots, k.$$

Remark 5.22

The third relation is known as Karush–Kuhn–Tucker complementarity condition. It implies that for active constraints, $*_i* \geq 0$, whereas for inactive constraints $_i = 0$. Furthermore, it is possible to show that for active constraints again changing the constraint to be b in place of 0, $\alpha_i^* = \left[\frac{\partial f^*}{\partial b_i}\right]_{b_i=0}$, so that the Lagrange multiplier represents the sensitivity of the optimal value to the constraint. Perturbing inactive constraints has no effect on the solution of the optimisation problem.

Remark 5.23

One way to interpret the above results is that a solution point can be in one of two positions with respect to an inequality constraint, either in the interior of the feasible region, with the constraint inactive, or on the boundary defined by that constraint with the constraint active. In the first case, the conditions for optimality for that constraint are given by Fermat's theorem, so the $_i$ need to be zero. In the second case, one can use Lagrange's theorem with a non-zero $_i$. So the KKT conditions say that either a constraint is active, meaning $g_i(\mathbf{w}^*) = 0$, or the corresponding multiplier satisfies $_i = 0$. This is summarised in the equation $g_i(\mathbf{w}^*)$ $*_i = 0$.

5.3 Duality

Lagrangian treatment of convex optimisation problems leads to an alternative dual description, which often turns out to be easier to solve than the primal problem since handling inequality constraints directly is difficult. The dual problem is obtained by introducing Lagrange multipliers, also called the dual variables. Dual methods are based on the idea that the dual variables are the fundamental unknowns of the problem.

We can transform the primal into a dual by simply setting to zero the derivatives of the Lagrangian with respect to the primal variables, and substituting the relations so obtained back into the Lagrangian, hence removing the dependence on the primal variables. This corresponds to explicitly computing the function

$$\theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \inf_{\mathbf{w} \in \Omega} L(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}).$$

The resulting function contains only dual variables and must be maximised under simpler constraints. This strategy will be adopted in subsequent chapters and has become one of the standard techniques in the theory of Support Vector Machines. The pleasing feature of the resulting expression for the primal variables is that it matches exactly the dual representation introduced in Chapter 2 and so will seem very natural in the context in which we will be using it. Hence, the use of dual representations in Support Vector Machines not only allows us to work in high dimensional spaces as indicated in Chapter 3, but also paves the way for algorithmic techniques derived from optimisation theory. As a further example the duality gap can be used as a convergence criterion for iterative techniques. A number of further consequences will flow from the convex quadratic programmes that arise in SVM optimisation. The Karush–Kuhn–Tucker complementarity conditions imply that only the active constraints will have non-zero dual variables, meaning that for certain optimisations the actual number of variables involved may be significantly fewer than the full training set size. We will see that the term *support vector* refers to those examples for which the dual variables are non-zero.

Example 5.24

(Quadratic programme) We demonstrate the practical use of duality by applying it to the important special case of a quadratic objective function,

$$\begin{aligned} &\text{minimise} && \tfrac{1}{2}\mathbf{w}'\mathbf{Q}\mathbf{w} - \mathbf{k}'\mathbf{w}, \\ &\text{subject to} && \mathbf{X}\mathbf{w} \leq \mathbf{c}, \end{aligned}$$

where $\mathbf{Q}$ is a positive definite $n \times n$ matrix, $\mathbf{k}$ is an n-vector; $\mathbf{c}$ an m-vector, $\mathbf{w}$ the unknown, and $\mathbf{X}$ an $m \times n$ matrix. Assuming that the feasible region is not empty, this problem can be rewritten as

$$\max_{\boldsymbol{\alpha} \geq 0} \left(\min_{\mathbf{w}} \left(\frac{1}{2}\mathbf{w}'\mathbf{Q}\mathbf{w} - \mathbf{k}'\mathbf{w} + \boldsymbol{\alpha}' (\mathbf{X}\mathbf{w} - \mathbf{c}) \right) \right).$$

The minimum over $\mathbf{w}$ is unconstrained, and is attained at $\mathbf{w} = \mathbf{Q}^{-1}(\mathbf{k} - \mathbf{X}$). Resubstituting this back in the original problem, one obtains the dual:

$$\begin{aligned} &\text{maximise} && -\tfrac{1}{2}\boldsymbol{\alpha}'\mathbf{P}\boldsymbol{\alpha} - \boldsymbol{\alpha}'\mathbf{d} - \tfrac{1}{2}\mathbf{k}'\mathbf{Q}\mathbf{k}, \\ &\text{subject to} && \boldsymbol{\alpha} \geq \mathbf{0}, \end{aligned}$$

where $\mathbf{P} = \mathbf{X}\mathbf{Q}^{-1}\mathbf{X}$, and $\mathbf{d} = \mathbf{c} - \mathbf{X}\mathbf{Q}^{-1}\mathbf{k}$. Thus, the dual of a quadratic program is another quadratic programme but with simpler constraints.

5.4 Exercises

1. A ball centred at a point $\mathbf{v}$ of radius R is the set

$$B_R(\mathbf{v}) = \{\mathbf{u} : \|\mathbf{u} - \mathbf{v}\| \leq R\}.$$

Express the problem of finding the ball of smallest radius that contains a given set

$$S = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$$

of vectors as an optimisation problem. See Figure 5.1 for a simple two dimensional example. Convert the problem derived to the dual form, hence showing that the solution can be expressed as a linear combination of the set S and can be solved in a kernel-induced feature space.

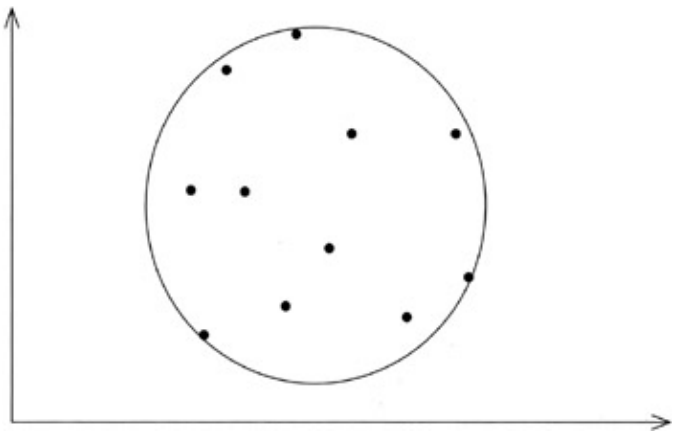


Figure 5.1: Example of a minimal enclosing sphere for a set of points in two dimensions

2. The convex hull of a set

$$T = \{\mathbf{x}_1, \ldots, \mathbf{x}_\ell\}$$

is the set of all convex combinations of the points in T. Given a linearly separable training set $S = S^+$ ∪ S^- of positive and negative examples, express the problem of finding points $\mathbf{x}^+$ and $\mathbf{x}^-$ in the convex hulls of S^+ and S^- for which the distance $\|\mathbf{x}^+ - \mathbf{x}^-\|$ is minimal as an optimisation problem. Note that this distance is twice the margin of the training set S.

3. Consider the parameter space of weight vectors for a linear learning machine. Each point of this space corresponds to one hypothesis for fixed bias. Each training example $\mathbf{x}$ gives rise to a hyperplane in this space defined by the equation

$$\langle \mathbf{w} \cdot \mathbf{x} \rangle = 0.$$

The situation for a two dimensional weight vector is shown in Figure 5.2 for three examples. Each hyperplane divides the space into those hypotheses giving the correct classification and those giving an incorrect classification. The region of hypotheses which correctly classify the whole training set is sometimes referred to as the version space. In Figure 5.2 this is the central triangle. Express the problem of finding the centre of the largest sphere completely contained in the version space, that is the point SV in Figure 5.2. Note that the point is distinct from the centre of mass of the version space also shown in the figure. Convert the problem to the dual form.

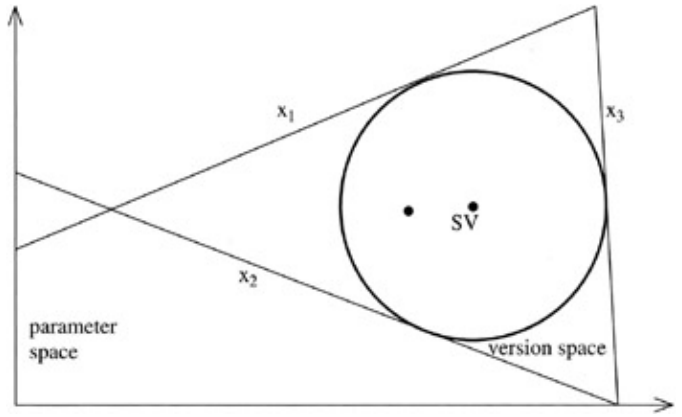


Figure 5.2: The version space for a linear learning machine

5.5 Further Reading and Advanced Topics

The theory of optimisation dates back to the work of Fermat, who formulated the result on stationarity for unconstrained problems in the 17th century. The extension to the constrained case was made by Lagrange in 1788, for the case of equality constraints. It was not until 1951 that the theory was generalised to the case of inequality constraints by Kuhn and Tucker [77], giving rise to the modern theory of convex optimisation. Karush had already described the optimality conditions in his dissertation in 1939 [69] and this is why the conditions arising from the Kuhn–Tucker theorem are usually referred to as the Karush–Kuhn–Tucker (KKT) conditions.

In the following years, considerable work was done by Wolfe, Mangasarian, Duran, and others, to extend the duality results known for linear programming to the convex programming case (see for example the introduction of [80]). The diffusion of computers in the 1960s greatly increased the interest in what was then known as mathematical programming, which studies the solution of problems by (usually linear or quadratic optimisation methods.

The use of optimisation in machine learning was pioneered by Olvi Mangasarian (for example, [84], [85], [87]) with his work on linear programming machines. See also [14] by Bennett et al. for a very nice discussion of linear and quadratic optimisation techniques applied to pattern recognition. Mangasarian took his ideas to their extreme, designing algorithms that can perform data mining on datasets of hundreds of thousands of points (see Section 7.8 for more references). The perceptron algorithm described in Chapter 2 can also be regarded as a simple optimisation procedure, searching for a feasible point given a set of linear constraints specified by the data. But rather than picking any point in the feasible region (an ill-posed problem) one could choose to pick some specific point satisfying some extremal property like the ones discussed in Chapter 4 such as being maximally distant from the boundaries of the feasible region. This type of consideration will lead to the development of Support Vector Machines in the next chapter. Mangasarian's early work mainly focused on minimising the 1-norm of the solution $\mathbf{w}$. Notice finally that the application of optimisation ideas to problems like least squares regression (discussed in Chapter 2) was already a use of such concepts in machine learning.

Optimisation theory is a well-developed and quite stable field, and the standard results summarised in this chapter can be found in any good textbook. A particularly readable and comprehensive text on optimisation theory is [11]; also the classic books [80], [41] and [86] provide good introductions. Optimisation theory usually also includes the algorithmic techniques needed to solve the problem practically. We will address this issue in Chapter 7. The important contribution of optimisation theory to the theory of Support Vector Machines, however, lies not on the algorithmic side, but rather in its providing a mathematical characterisation of the solutions, via the KKT conditions, giving a mathematical meaning to the dual variables (introduced in Chapter 2) and to the margin slack vector (introduced in Chapter 4), and generally in providing a geometrical intuition for the dual problem.

Exercises 1 concerning the centre of the smallest ball containing the data is relevant if we wish to minimise the estimate of the fat shattering dimension of a given set of points by optimally shifting the origin. This problem was first studied in [128], while Exercise 2 concerning the distance between convex hulls is discussed in [14], and provides an efficient way to characterise the maximal margin hyperplane. This is discussed in [73] where it is used to motivate an interesting algorithm (see Section 7.8 for more references).

These references are also given on the website **http://www.support-vector.net**, which will be kept up to date with new work, pointers to software and papers that are available on-line.

Chapter 6: Support Vector Machines

The material covered in the first five chapters has given us the foundation on which to introduce Support Vector Machines, the learning approach originally developed by Vapnik and co-workers. Support Vector Machines are a system for efficiently training the linear learning machines introduced in Chapter 2 in the kernel-induced feature spaces described in Chapter 3, while respecting the insights provided by the generalisation theory of Chapter 4, and exploiting the optimisation theory of Chapter 5. An important feature of these systems is that, while enforcing the learning biases suggested by the generalisation theory, they also produce ‘sparse’ dual representations of the hypothesis, resulting in extremely efficient algorithms. This is due to the Karush–Kuhn–Tucker conditions, which hold for the solution and play a crucial role in the practical implementation and analysis of these machines. Another important feature of the Support Vector approach is that due to Mercer's conditions on the kernels the corresponding optimisation problems are convex and hence have no local minima. This fact, and the reduced number of non-zero parameters, mark a clear distinction between these system and other pattern recognition algorithms, such as neural networks. This chapter will also describe the optimisation required to implement the Bayesian learning strategy using Gaussian processes.

6.1 Support Vector Classification

The aim of Support Vector classification is to devise a computationally efficient way of learning ‘good’ separating hyperplanes in a high dimensional feature space, where by ‘good’ hyperplanes we will understand ones optimising the generalisation bounds described in Chapter 4, and by ‘computationally efficient’ we will mean algorithms able to deal with sample sizes of the order of 100 000 instances. The generalisation theory gives clear guidance about how to control capacity and hence prevent overfitting by controlling the hyperplane margin measures, while optimisation theory provides the mathematical techniques necessary to find hyperplanes optimising these measures. Different generalisation bounds exist, motivating different algorithms: one can for example optimise the maximal margin, the margin distribution, the number of support vectors, etc. This chapter will consider the most common and well-established approaches which reduce the problem to minimising the norm of the weight vector. At the end of the chapter we will provide pointers to other related algorithms, though since research in this field is still in progress, we make no attempt to be exhaustive.

6.1.1 The Maximal Margin Classifier

The simplest model of Support Vector Machine, which was also the first to be introduced, is the so-called maximal margin classifier. It works only for data which are linearly separable in the feature space, and hence cannot be used in many real-world situations. Nonetheless it is the easiest algorithm to understand, and it forms the main building block for the more complex Support Vector Machines. It exhibits the key features that characterise this kind of learning machine, and its description is therefore crucial for understanding the more advanced systems introduced later.

Theorem 4.18 in Chapter 4 bounds the generalisation error of linear machines in terms of the margin $m_S(f)$ of the hypothesis f with respect to the training set S. The dimensionality of the space in which the data are separated does not appear in this theorem. The maximal margin classifier optimises this bound by separating the data with the maximal margin hyperplane, and given that the bound does not depend on the dimensionality of the space, this separation can be sought in any kernel-induced feature space. The maximal margin classifier forms the strategy of the first Support Vector Machine, namely to find the maximal margin hyperplane in an appropriately chosen kernel-induced feature space.

This strategy is implemented by reducing it to a convex optimisation problem: minimising a quadratic function under linear inequality constraints. First we note that in the definition of linear classifiers there is an

inherent degree of freedom, due to the fact that the function associated with the hyperplane ($\mathbf{w}$, b) does not change if we rescale the hyperplane to ($\mathbf{w}$, b), for $\mathbb{R}^+$. There will, however, be a change in the margin as measured by the function output as opposed to the geometric margin. We refer to the margin of the function output as the *functional margin*. Theorem 4.18 involves the *geometric margin*, that is the functional margin of a normalised weight vector. Hence, we can equally well optimise the geometric margin by fixing the functional margin to be equal to 1 (hyperplanes with functional margin 1 are sometimes known as *canonical hyperplanes*) and minimising the norm of the weight vector. If $\mathbf{w}$ is the weight vector realising a functional margin of 1 on the positive point $\mathbf{x}^+$ and the negative point $\mathbf{x}^-$, we can compute its geometric margin as follows. Recall that a functional margin of 1 implies

$$\begin{aligned}\langle \mathbf{w} \cdot \mathbf{x}^+ \rangle + b &= +1, \\ \langle \mathbf{w} \cdot \mathbf{x}^- \rangle + b &= -1,\end{aligned}$$

while to compute the geometric margin we must normalise $\mathbf{w}$. The geometric margin is then the functional margin of the resulting classifier

$$\begin{aligned}\gamma &= \frac{1}{2}\left(\left\langle \frac{\mathbf{w}}{\|\mathbf{w}\|_2} \cdot \mathbf{x}^+ \right\rangle - \left\langle \frac{\mathbf{w}}{\|\mathbf{w}\|_2} \cdot \mathbf{x}^- \right\rangle\right) \\ &= \frac{1}{2\|\mathbf{w}\|_2}\left(\langle \mathbf{w} \cdot \mathbf{x}^+ \rangle - \langle \mathbf{w} \cdot \mathbf{x}^- \rangle\right) \\ &= \frac{1}{\|\mathbf{w}\|_2}.\end{aligned}$$

Hence, the resulting geometric margin will be equal to $1/\|\mathbf{w}\|_2$ and we have demonstrated the following result.

Proposition 6.1

Given a linearly separable training sample

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$$

the hyperplane ($\mathbf{w}$, b) that solves the optimisation problem

$$\begin{aligned}&\textit{minimise}_{\mathbf{w},b} && \langle \mathbf{w} \cdot \mathbf{w} \rangle, \\ &\textit{subject to} && y_i\left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b\right) \geq 1, \\ & && i = 1, \ldots, \ell,\end{aligned}$$

realises the maximal margin hyperplane with geometric margin $\gamma = 1/\|\mathbf{w}\|_2$.

We now consider how to transform this optimisation problem into its corresponding dual problem following the strategy outlined in Section 5.3. The primal Lagrangian is

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle - \sum_{i=1}^{\ell} \alpha_i \left[y_i\left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b\right) - 1\right]$$

where $\alpha_i \geq 0$ are the Lagrange multipliers, as described in Chapter 5.

The corresponding dual is found by differentiating with respect to $\mathbf{w}$ and b, imposing stationarity,

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{\ell} y_i \alpha_i \mathbf{x}_i = \mathbf{0},$$

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial b} = \sum_{i=1}^{\ell} y_i \alpha_i = 0,$$

and resubstituting the relations obtained,

$$\mathbf{w} = \sum_{i=1}^{\ell} y_i \alpha_i \mathbf{x}_i,$$

$$0 = \sum_{i=1}^{\ell} y_i \alpha_i,$$

into the primal to obtain

$$\begin{aligned} L(\mathbf{w}, b, \boldsymbol{\alpha}) &= \frac{1}{2} \langle \mathbf{w} \cdot \mathbf{w} \rangle - \sum_{i=1}^{\ell} \alpha_i \left[y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) - 1 \right] \\ &= \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle - \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle + \sum_{i=1}^{\ell} \alpha_i \\ &= \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle . \end{aligned}$$

Remark 6.2

The first of the substitution relations shows that the hypothesis can be described as a linear combination of the training points: the application of optimisation theory naturally leads to the dual representation introduced in Chapter 2. The dual representation is also required for the use of kernels.

We have therefore shown the main part of the following proposition, which follows from Proposition 6.1.

Proposition 6.3

Consider a linearly separable training sample

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)),$$

and suppose the parameters $\boldsymbol{\alpha}^*$ solve the following quadratic optimisation problem:

$$\begin{aligned} &\textit{maximise} && W(\boldsymbol{\alpha}) = \textstyle\sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle, \\ &\textit{subject to} && \textstyle\sum_{i=1}^{\ell} y_i \alpha_i = 0, \\ & && \alpha_i \geq 0, \; i = 1, \ldots, \ell. \end{aligned} \qquad (6.1)$$

Then the weight vector $\mathbf{w}^* = \sum_{i=1} y_i \alpha_i^* \mathbf{x}_i$ realises the maximal margin hyperplane with geometric margin

$$\gamma = 1 / \|\mathbf{w}^*\|_2 .$$

Remark 6.4

The value of b does not appear in the dual problem and so b^* must be found making use of the primal constraints:

$$b^* = -\frac{\max_{y_i=-1} \left(\langle \mathbf{w}^* \cdot \mathbf{x}_i \rangle \right) + \min_{y_i=1} \left(\langle \mathbf{w}^* \cdot \mathbf{x}_i \rangle \right)}{2}$$

Theorem 5.21 in Chapter 5 applies to this optimisation problem. The Karush–Kuhn–Tucker complementarity conditions provide useful information about the structure of the solution. The conditions state that the optimal solutions $\boldsymbol{\alpha}^*$, ($\mathbf{w}^*$, b^*) must satisfy

$$\alpha_i^* \left[y_i \left(\langle \mathbf{w}^* \cdot \mathbf{x}_i \rangle + b^* \right) - 1 \right] = 0, \qquad i = 1, \ldots, \ell.$$

This implies that only for inputs $\mathbf{x}_i$ for which the functional margin is one and that therefore lie closest to the hyperplane are the corresponding α_i^* non-zero. All the other parameters α_i^* are zero. Hence, in the expression for the weight vector only these points are involved. It is for this reason that they are called *support vectors*, see Figure 6.1 We will denote the set of indices of the support vectors with sv.

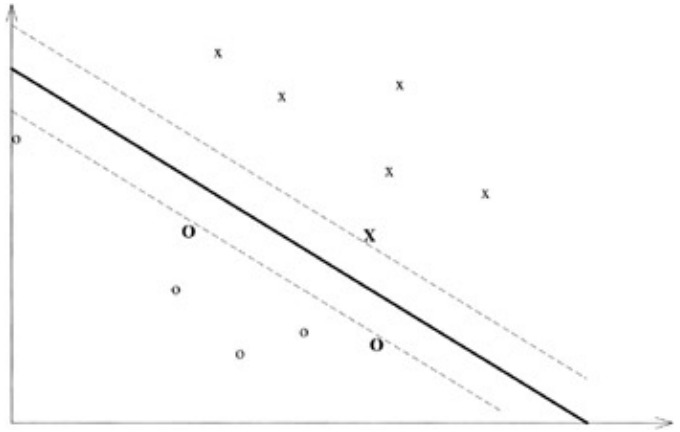


Figure 6.1: A maximal margin hyperplane with its support vectors highlighted

Furthermore the optimal hyperplane can be expressed in the dual representation in terms of this subset of the parameters:

$$\begin{aligned} f(\mathbf{x}, \boldsymbol{\alpha}^*, b^*) &= \sum_{i=1}^{\ell} y_i \alpha_i^* \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b^* \\ &= \sum_{i \in \mathrm{sv}} y_i \alpha_i^* \langle \mathbf{x}_i \cdot \mathbf{x} \rangle + b^*. \end{aligned}$$

The Lagrange multipliers associated with each point become the dual variables, giving them an intuitive interpretation quantifying how important a given training point is in forming the final solution. Points that are not support vectors have no influence, so that in non-degenerate cases slight perturbations of such points will not affect the solution. A similar meaning was found in the case of the dual representations for the perceptron learning algorithm, where the dual variable was proportional to the number of mistakes made by the hypothesis on a given point during the training.

Another important consequence of the Karush-Kuhn-Tucker complementarity conditions is that for j sv,

$$y_j f(\mathbf{x}_j, \boldsymbol{\alpha}^*, b^*) = y_j \left(\sum_{i \in \mathrm{sv}} y_i \alpha_i^* \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle + b^* \right) = 1,$$

and therefore

$$\begin{aligned}\langle \mathbf{w}^* \cdot \mathbf{w}^* \rangle &= \sum_{i,j=1}^{\ell} y_i y_j \alpha_i^* \alpha_j^* \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle \\ &= \sum_{j \in sv} \alpha_j^* y_j \sum_{i \in sv} y_i \alpha_i^* \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle \\ &= \sum_{j \in sv} \alpha_j^* \left(1 - y_j b^*\right) \\ &= \sum_{i \in sv} \alpha_i^* .\end{aligned}$$

We therefore have the following proposition.

Proposition 6.5

Consider a linearly separable training sample

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)),$$

and suppose the parameters α^* and b^* solve the dual optimisation problem (6.1). Then the weight vector $\mathbf{w} = \sum_{i=1} y_i \alpha_i^* \mathbf{x}_i$ realises the maximal margin hyperplane with geometric margin

$$\gamma = 1/\|\mathbf{w}\|_2 = \left(\sum_{i \in sv} \alpha_i^* \right)^{-1/2} .$$

Both the dual objective and the decision function have the remarkable property that the data only appear inside an inner product. This will make it possible to find and use optimal hyperplanes in the feature space through the use of kernels as shown in the following proposition.

Proposition 6.6

Consider a training sample

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$$

that is linearly separable in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$ and suppose the parameters α^* and b^* solve the following quadratic optimisation problem:

$$\begin{aligned} &\textit{maximise} && W(\boldsymbol{\alpha}) = \textstyle\sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \\ &\textit{subject to} && \textstyle\sum_{i=1}^{\ell} y_i \alpha_i = 0, \\ & && \alpha_i \geq 0,\ i = 1, \ldots, \ell. \end{aligned} \qquad (6.2)$$

Then the decision rule given by $\mathrm{sgn}(f(\mathbf{x}))$, where $f(\mathbf{x}) = \sum_{i=1} y_i \alpha_i^* \boldsymbol{K}(\mathbf{x}_i, \mathbf{x}) + b^*$, is equivalent to the maximal margin hyperplane in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$ and that hyperplane has geometric margin

$$\gamma = \left(\sum_{i \in sv} \alpha_i^* \right)^{-1/2} .$$

Note that the requirement that the kernel satisfy Mercer's conditions is equivalent to the requirement that the matrix with entries $(K(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1}$ be positive definite for all training sets. This in turn means that the optimisation problem (6.2) is convex since the matrix $(y_i y_j K(\mathbf{x}_i, \mathbf{x}_j))_{i,j=1}$ is also positive definite. Hence, the

property required for a kernel function to define a feature space also ensures that the maximal margin optimisation problem has a unique solution that can be found efficiently. This rules out the problem of local minima encountered in training neural networks.

Remark 6.7

The maximal margin classifier is motivated by Theorem 4.18 which bounds the generalisation error in terms of the margin and the radius of a ball centred at the origin containing the data. One advantage of motivating an algorithm using such a theorem is that we can compute a bound on the generalisation as an output of the learning algorithm. The value of the margin is given in Proposition 6.6, while the radius of the ball centred at the origin in feature space can be computed as

$$R = \max_{1 \leq i \leq \ell} \left(K(\mathbf{x}_i, \mathbf{x}_i) \right).$$

Unfortunately, though the strategy suggested by the theorem has been shown to be very effective, the constants involved typically make the actual value of the resulting bound unrealistic. There is still, however, the potential for using the bounds to choose between for example different kernels, where it is the relative size that is important, though the accuracy of the bounds is still the subject of active research.

An important result from the optimisation theory chapter is Theorem 5.15 stating that the primal objective is always bigger than the dual objective. Since the problem we are considering satisfies the conditions of Theorem 5.20, there is no duality gap at the optimal solution. We can therefore use any difference between the primal and dual values as an indicator of convergence. We will call this difference the feasibility gap. Let $\hat{\boldsymbol{\alpha}}$ be the current value of the dual variables. The weight vector is calculated from setting the derivative of the Lagrangian equal to zero, and so the current value of the weight vector $\hat{\mathbf{w}}$ is the one which minimises $L(\mathbf{w}, b, \hat{\boldsymbol{\alpha}})$ for the given $\hat{\boldsymbol{\alpha}}$. Hence, this difference can be computed as follows:

$$\begin{aligned} W(\hat{\boldsymbol{\alpha}}) - \frac{1}{2}\|\hat{\mathbf{w}}\|^2 &= \inf_{\mathbf{w},b} L(\mathbf{w}, b, \hat{\boldsymbol{\alpha}}) - \frac{1}{2}\|\hat{\mathbf{w}}\|^2 \\ &= L(\hat{\mathbf{w}}, b, \hat{\boldsymbol{\alpha}}) - \frac{1}{2}\|\hat{\mathbf{w}}\|^2 \\ &= -\sum_{i=1}^{\ell} \hat{\alpha}_i \left[y_i \left(\langle \hat{\mathbf{w}} \cdot \mathbf{x}_i \rangle + b \right) - 1 \right] \\ &= \sum_{i=1}^{\ell} \hat{\alpha}_i - \sum_{i,j=1}^{\ell} \hat{\alpha}_i y_i y_j \hat{\alpha}_j \langle \mathbf{x}_j \cdot \mathbf{x}_i \rangle, \end{aligned}$$

which is minus the sum of the Karush–Kuhn–Tucker complementarity conditions. Note that this will correspond to the difference between primal and dual feasible solutions provided $\hat{\mathbf{w}}$ satisfies the primal constraints, that is provided $y_i \left(\langle \hat{\mathbf{w}} \cdot \mathbf{x}_i \rangle + b \right) \geq 1$ for all i, which is equivalent to

$$y_i \left(\sum_{j=1}^{\ell} y_j \hat{\alpha}_j \langle \mathbf{x}_j \cdot \mathbf{x}_i \rangle + b \right) \geq 1.$$

There is no guarantee that this will hold, and so computation of a feasibility gap is not straightforward in the maximal margin case. We will see below that for one of the soft margin cases we can estimate the feasibility gap.

The fact that only a subset of the Lagrange multipliers is non-zero is referred to as *sparseness*, and means that the support vectors contain all the information necessary to reconstruct the hyperplane. Even if all of the other points were removed the same maximal separating hyperplane would be found for the remaining subset of the

support vectors. This can also be seen from the dual problem, since removing rows and columns corresponding to non-support vectors leaves the same optimisation problem for the remaining submatrix. Hence, the optimal solution remains unchanged. This shows that the maximal margin hyperplane is a compression scheme according to the definition of Section 4.4, since given the subset of support vectors we can reconstruct the maximal margin hyperplane that will correctly classify the whole training set. Applying Theorem 4.25, we obtain the following result.

Theorem 6.8

Consider thresholding real-valued linear functions with unit weight vectors on an inner product space X. For any probability distribution 풟 on $X \times \{-1, 1\}$, with probability $1 -$ over random examples-S, the maximal margin hyperplane has error no more than

$$\operatorname{err}_{\mathcal{D}}(f) \leq \frac{1}{\ell - d}\left(d \log \frac{e\ell}{d} + \log \frac{\ell}{\delta}\right),$$

where $d = \#\,\mathrm{sv}$ is the number of support vectors.

The theorem shows that the fewer the number of support vectors the better generalisation can be expected. This is closely related to the Ockham approach of finding a compact representation of the classification function. The nice property of the bound is that it does not depend explicitly on the dimension of the feature space.

Remark 6.9

A slightly tighter bound on the *expected* generalisation error in terms of the same quantities can be obtained by a leave-one-out argument. Since, when a non-support vector is omitted, it is correctly classified by the remaining subset of the training data the leave-one-out estimate of the generalisation error is

$$\frac{\#\,\mathrm{sv}}{\ell}.$$

A cyclic permutation of the training set shows that the expected error of a test point is bounded by this quantity. The use of an expected generalisation bound gives no guarantee about its variance and hence its reliability. Indeed leave-one-out bounds are known to suffer from this problem. Theorem 6.8 can be seen as showing that in the case of maximal margin classifiers an only very slightly weaker bound does hold with high probability and hence that in this case the variance cannot be too high.

The maximal margin classifier does not attempt to control the number of support vectors and yet in practice there are frequently very few support vectors. This sparseness of the solution will also motivate a number of implementation techniques for dealing with large datasets, which we will discuss in more detail in Chapter 7.

The only degree of freedom in the maximal margin algorithm is the choice of kernel, which amounts to model selection. Any prior knowledge we have of the problem can help in choosing a parametrised kernel family, and then model selection is reduced to adjusting the parameters. For most classes of kernels, for example polynomial or Gaussian, it is always possible to find a kernel parameter for which the data become separable. In general, however, forcing separation of the data can easily lead to overfitting, particularly when noise is present in the data.

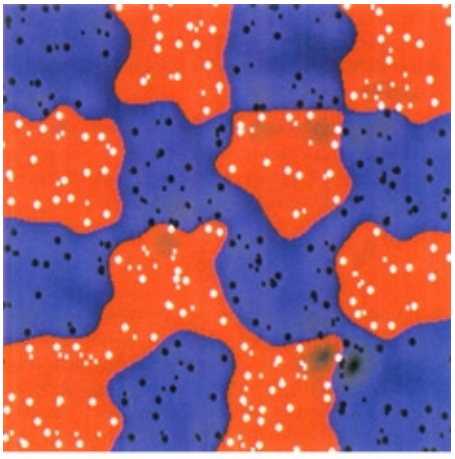
In this case, outliers would typically be characterised by a large Lagrange multiplier, and the procedure could be used for data cleaning, since it can rank the training data according to how difficult they are to classify correctly.

This algorithm provides the starting point for the many variations on this theme proposed in the last few years

and attempting to address some of its weaknesses: that it is sensitive to the presence of noise; that it only considers two classes; that it is not expressly designed to achieve sparse solutions.

Remark 6.10

Note that in SVMs the margin has two effects. On the one hand, its maximisation ensures low fat-shattering dimension, and hence better generalisation, while on the other hand the margin is the origin of the sparseness of the solution vector, as the inequality constraints generate the Karush-Kuhn-Tucker complementarity conditions.



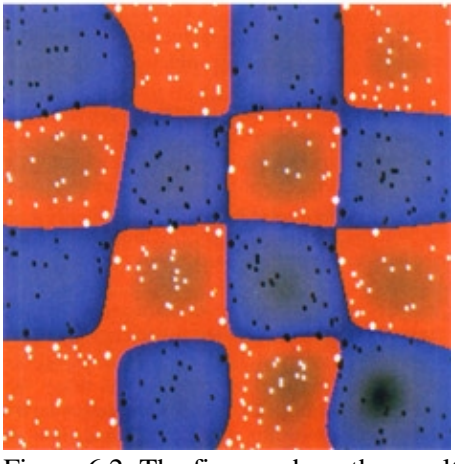


Figure 6.2: The figures show the result of the maximum margin SVM for learning a chess board from points generated according to the uniform distribution using Gaussian kernels with different values of . The white dots are the positive points and the black dots the negative ones. The support vectors are indicated by large dots. The red area comprises those points that are positively classified by the decision function, while the area classified negative is coloured blue. Notice that in both cases the classification of the training set is consistent. The size of the functional margin is indicated by the level of shading. The images make clear how the accuracy of the resulting classifier can be affected by the choice of kernel parameter. In image (b) with the large value of , each region has only a small number of support vectors and the darker shading clearly indicates where the machine has more confidence of its classification. In contast image (a) has a more complex boundary, significantly more support vectors, and there are very few regions with darker shading

6.1.2 Soft Margin Optimisation

The maximal margin classifier is an important concept, as a starting point for the analysis and construction of more sophisticated Support Vector Machines, but it cannot be used in many real-world problems (we will see an exception in Chapter 8): if the data are noisy, there will in general be no linear separation in the feature space (unless we are ready to use very powerful kernels, and hence overfit the data). The main problem with

the maximal margin classifier is that it always produces perfectly a consistent hypothesis, that is a hypothesis with no training error. This is of course a result of its motivation in terms of a bound that depends on the margin, a quantity that is negative unless the data are perfectly separated.

The dependence on a quantity like the margin opens the system up to the danger of falling hostage to the idiosyncrasies of a few points. In real data, where noise can always be present, this can result in a brittle estimator. Furthermore, in the cases where the data are not linearly separable in the feature space, the optimisation problem cannot be solved as the primal has an empty feasible region and the dual an unbounded objective function. These problems motivate using the more robust measures of the *margin distribution* introduced in Chapter 4. Such measures can tolerate noise and outliers, and take into consideration the positions of more training points than just those closest to the boundary.

Recall that the primal optimisation problem for the maximal margin case is the following:

$$\begin{array}{ll}\text{minimise}_{\mathbf{w},b} & \langle \mathbf{w} \cdot \mathbf{w} \rangle, \\ \text{subject to} & y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) \geq 1, \, i = 1, \ldots, \ell. \end{array}$$

In order to optimise the margin slack vector we need to introduce slack variables to allow the margin constraints to be violated

$$\begin{array}{ll} \text{subject to} & y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) \geq 1 - \xi_i, \, i = 1, \ldots, \ell, \\ & \xi_i \geq 0, \, i = 1, \ldots, \ell. \end{array}$$

Theorem 4.22 in Chapter 4 bounds the generalisation error in terms of the 2-norm of the margin slack vector, the so-called 2-norm soft margin, which contains the ξ_i scaled by the norm of the weight vector **w.** Hence, the equivalent expression on which the generalisation depends is

$$\begin{aligned} \frac{R^2 + \frac{\|\xi\|_2^2}{\|\mathbf{w}\|_2^2}}{\gamma^2} &= \|\mathbf{w}\|_2^2 \left(R^2 + \frac{\|\xi\|_2^2}{\|\mathbf{w}\|_2^2} \right) \\ &= \|\mathbf{w}\|_2^2 R^2 + \|\xi\|_2^2, \end{aligned}$$

suggesting that an optimal choice for C in the objective function of the resulting optimisation problem should be R^{-2}:

$$\begin{array}{ll}\text{minimise}_{\xi,\mathbf{w},b} & \langle \mathbf{w} \cdot \mathbf{w} \rangle + C \sum_{i=1}^{\ell} \xi_i^2 \\ \text{subject to} & y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) \geq 1 - \xi_i, \, i = 1, \ldots, \ell, \\ & \xi_i \geq 0, \, i = 1, \ldots, \ell. \end{array} \qquad (6.3)$$

Notice that if $\xi_i < 0$, then the first constraint will still hold if we set $\xi_i = 0$, while this change will reduce the value of the objective function. Hence, the optimal solution for the problem obtained by removing the positivity constraint on ξ_i will coincide with the optimal solution of equation (6.3). Hence we obtain the solution to equation (6.3) by solving the following optimisation problem:

$$\begin{array}{ll}\text{minimise}_{\xi,\mathbf{w},b} & \langle \mathbf{w} \cdot \mathbf{w} \rangle + C \sum_{i=1}^{\ell} \xi_i^2, \\ \text{subject to} & y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) \geq 1 - \xi_i, \, i = 1, \ldots, \ell, \end{array} \qquad (6.4)$$

In practice the parameter C is varied through a wide range of values and the optimal performance assessed using a separate validation set or a technique known as cross-validation for verifying performance using only the training set. As the parameter C runs through a range of values, the norm $\|\mathbf{w}\|_2$ varies smoothly through a corresponding range. Hence, for a particular problem, choosing a particular value for C corresponds to choosing a value for $\|\mathbf{w}\|_2$, and then minimising $\| \ \|_2$ for that size of **w.** This approach is also adopted in the 1-norm case where the optimisation problem minimises a combination of the norm of the weights and the 1-norm of the slack variables that does not exactly match that found in Theorem 4.24:

$$\begin{array}{ll}\text{minimise}_{\xi,w,b} & \langle \mathbf{w}\cdot\mathbf{w}\rangle + C\sum_{i=1}^{\ell}\xi_i, \\ \text{subject to} & y_i\left(\langle \mathbf{w}\cdot\mathbf{x}_i\rangle + b\right) \geq 1-\xi_i,\ i=1,\ldots,\ell, \\ & \xi_i \geq 0,\ i=1,\ldots,\ell.\end{array} \qquad (6.5)$$

Since there is a value of C corresponding to the optimal choice of $\|\mathbf{w}\|_2$, that value of C will give the optimal bound as it will correspond to finding the minimum of $\|\ \|_1$ with the given value for $\|\mathbf{w}\|_2$.

We will devote the next two subsubsections to investigating the duals of the two margin slack vector problems creating the so-called soft margin algorithms.

2-Norm Soft Margin – Weighting the Diagonal

The primal Lagrangian for the problem of equation (6.4) is

$$L(\mathbf{w},b,\xi,\alpha) = \frac{1}{2}\langle \mathbf{w}\cdot\mathbf{w}\rangle + \frac{C}{2}\sum_{i=1}^{\ell}\xi_i^2 - \sum_{i=1}^{\ell}\alpha_i\left[y_i\left(\langle \mathbf{w}\cdot\mathbf{x}_i\rangle + b\right) - 1 + \xi_i\right]$$

where $_i \geq 0$ are the Lagrange multipliers, as described in Chapter 5.

The corresponding dual is found by differentiating with respect to $\mathbf{w}$, and b, imposing stationarity,

$$\begin{aligned}\frac{\partial L(\mathbf{w},b,\xi,\alpha)}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{\ell} y_i\alpha_i\mathbf{x}_i = \mathbf{0}, \\ \frac{\partial L(\mathbf{w},b,\xi,\alpha)}{\partial \xi} &= C\xi - \alpha = \mathbf{0}, \\ \frac{\partial L(\mathbf{w},b,\xi,\alpha)}{\partial b} &= \sum_{i=1}^{\ell} y_i\alpha_i = 0,\end{aligned}$$

and resubstituting the relations obtained into the primal to obtain the following adaptation of the dual objective function:

$$\begin{aligned}L(\mathbf{w},b,\xi,\alpha) &= \sum_{i=1}^{\ell}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{\ell} y_iy_j\alpha_i\alpha_j\langle \mathbf{x}_i\cdot\mathbf{x}_j\rangle + \frac{1}{2C}\langle \alpha\cdot\alpha\rangle - \frac{1}{C}\langle \alpha\cdot\alpha\rangle \\ &= \sum_{i=1}^{\ell}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{\ell} y_iy_j\alpha_i\alpha_j\langle \mathbf{x}_i\cdot\mathbf{x}_j\rangle - \frac{1}{2C}\langle \alpha\cdot\alpha\rangle.\end{aligned}$$

Hence, maximising the above objective over is equivalent to maximising

$$W(\alpha) = \sum_{i=1}^{\ell}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{\ell} y_iy_j\alpha_i\alpha_j\left(\langle \mathbf{x}_i\cdot\mathbf{x}_j\rangle + \frac{1}{C}\delta_{ij}\right),$$

where $_{ij}$ is the Kronecker defined to be 1 if $i = j$ and 0. The corresponding Karush–Kuhn–Tucker complementarity conditions are

$$\alpha_i\left[y_i(\langle \mathbf{x}_i\cdot\mathbf{w}\rangle + b) - 1 + \xi_i\right] = 0, \quad i=1,\ldots,\ell.$$

Hence, we have the following result in which we have moved directly to the more general kernel version.

Proposition 6.11

Consider classifying a training sample

$$S = ((\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_\ell,y_\ell)),$$

using the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$, and suppose the parameters α^* solve the following quadratic optimisation problem:

$$\begin{aligned} &\textit{maximise} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j \left(K(\mathbf{x}_i, \mathbf{x}_j) + \tfrac{1}{C}\delta_{ij}\right), \\ &\textit{subject to} \quad \sum_{i=1}^{\ell} y_i \alpha_i = 0, \\ &\qquad\qquad \alpha_i \geq 0,\ i = 1, \ldots, \ell. \end{aligned}$$

Let $f(\mathbf{x}) = \sum_{i=1} y_i \alpha_i^* K(\mathbf{x}_i, \mathbf{x}) + b^*$, where b^* is chosen so that $y_i f(\mathbf{x}_i) = 1 - \alpha_i^*/C$ for any i with $\alpha_i^* \neq 0$. Then the decision rule given by $\mathrm{sgn}(f(\mathbf{x}))$ is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$ which solves the optimisation problem (6.3), where the slack variables are defined relative to the geometric margin

$$\gamma = \left(\sum_{i \in sv} \alpha_i^* - \frac{1}{C} \langle \boldsymbol{\alpha}^* \cdot \boldsymbol{\alpha}^* \rangle \right)^{-1/2}.$$

Proof The value of b^* is chosen using the relation $\alpha_i = C \alpha_i$ and by reference to the primal constraints which by the Karush–Kuhn–Tucker complementarity conditions

$$\alpha_i \left[y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) - 1 + \xi_i \right] = 0,\ i = 1, \ldots, \ell,$$

must be equalities for non-zero α_i. It remains to compute the norm of $\mathbf{w}^*$ which defines the size of the geometric margin.

$$\begin{aligned} \langle \mathbf{w}^* \cdot \mathbf{w}^* \rangle &= \sum_{i,j=1}^{\ell} y_i y_j \alpha_i^* \alpha_j^* K(\mathbf{x}_i, \mathbf{x}_j) \\ &= \sum_{j \in sv} \alpha_j^* y_j \sum_{i \in sv} y_i \alpha_i^* K(\mathbf{x}_i, \mathbf{x}_j) \\ &= \sum_{j \in sv} \alpha_j^* \left(1 - \xi_j^* - y_j b^*\right) \\ &= \sum_{i \in sv} \alpha_i^* - \sum_{i \in sv} \alpha_i^* \xi_i^* \\ &= \sum_{i \in sv} \alpha_i^* - \frac{1}{C} \langle \boldsymbol{\alpha}^* \cdot \boldsymbol{\alpha}^* \rangle. \end{aligned}$$

This is still a quadratic programming problem, and can be used with the same methods as used for the maximal margin hyperplane. The only change is the addition of $1/C$ to the diagonal of the inner product matrix associated with the training set. This has the effect of adding $1/C$ to the eigenvalues of the matrix, rendering the problem better conditioned. We can therefore view the new problem as simply a change of kernel

$$K'(\mathbf{x}, \mathbf{z}) = K(\mathbf{x}, \mathbf{z}) + \frac{1}{C} \delta_{\mathbf{x}}(\mathbf{z}).$$

1-Norm Soft Margin – the Box Constraint

The corresponding Lagrangian for the 1-norm soft margin optimisation problem is

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r}) = \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle + C \sum_{i=1}^{\ell} \xi_i - \sum_{i=1}^{\ell} \alpha_i \left[y_i (\langle \mathbf{x}_i \cdot \mathbf{w} \rangle + b) - 1 + \xi_i \right] - \sum_{i=1}^{\ell} r_i \xi_i$$

with $_i \geq 0$ and $r_i \geq 0$. The corresponding dual is found by differentiating with respect to **w,** and b, imposing stationarity,

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{\ell} y_i \alpha_i \mathbf{x}_i = \mathbf{0},$$

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r})}{\partial \xi_i} = C - \alpha_i - r_i = 0,$$

$$\frac{\partial L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r})}{\partial b} = \sum_{i=1}^{\ell} y_i \alpha_i = 0,$$

and resubstituting the relations obtained into the primal; we obtain the following adaptation of the dual objective function:

$$L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle,$$

which curiously is identical to that for the maximal margin. The only difference is that the constraint $C - _i - r_i = 0$, together with $r_i \geq 0$, enforces $_i \leq C$, while $_i \neq 0$ only if $r_i = 0$ and therefore $_i = C$. The Karush–Kuhn–Tucker complementarity conditions are therefore

$$\alpha_i \left[y_i (\langle \mathbf{x}_i \cdot \mathbf{w} \rangle + b) - 1 + \xi_i \right] = 0, \quad i = 1, \ldots, \ell,$$
$$\xi_i (\alpha_i - C) = 0, \quad i = 1, \ldots, \ell.$$

Notice that the KKT conditions implies that non-zero slack variables can only occur when $_i = C$. The points with non-zero slack variables are 1/ ||**w**||-margin errors, as their geometric margin is less than 1/ ||**w**||. Points for which $0 < _i < C$ lie at the target distance of 1/ ||**w**|| from the hyperplane. We therefore have the following proposition.

Proposition 6.12

Consider classifying a training sample

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)),$$

using the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$, and suppose the parameters * solve the following quadratic optimisation problem:

$$\begin{array}{ll} \textit{maximise} & W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \\ \textit{subject to} & \sum_{i=1}^{\ell} y_i \alpha_i = 0, \\ & C \geq \alpha_i \geq 0, \; i = 1, \ldots, \ell. \end{array} \qquad (6.6)$$

Let $f(\mathbf{x}) = \sum_{i=1} \; y_i \; _i{}^* K(\mathbf{x}_i, \mathbf{x}) + b^*$, where b^* is chosen so that $y_i f(\mathbf{x}_i) = 1$ for any i with $C > _i{}^* > 0$. Then the decision rule given by sgn($f(\mathbf{x})$) is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$ that solves the optimisation problem (6.5), where the slack variables are defined relative to the geometric margin

$$\gamma = \left(\sum_{i,j \in sv} y_i y_j \alpha_i^* \alpha_j^* K(\mathbf{x}_i, \mathbf{x}_j) \right)^{-1/2}.$$

Proof The value of *b** is chosen using the Karush–Kuhn–Tucker complementarity conditions which imply that if $C > \alpha_i^* > 0$ both $\alpha_i^* = 0$ and

$$y_i(\langle \mathbf{x}_i \cdot \mathbf{w}^* \rangle + b^*) - 1 + \xi_i^* = 0.$$

The norm of **w*** is clearly given by the expression

$$\begin{aligned} \langle \mathbf{w}^* \cdot \mathbf{w}^* \rangle &= \sum_{i,j=1}^{\ell} y_i y_j \alpha_i^* \alpha_j^* K(\mathbf{x}_i, \mathbf{x}_j) \\ &= \sum_{j \in sv} \sum_{i \in sv} y_i y_j \alpha_i^* \alpha_j^* K(\mathbf{x}_i, \mathbf{x}_j). \end{aligned}$$

So surprisingly this problem is equivalent to the maximal margin hyperplane, with the additional constraint that all the α_i are upper bounded by *C*. This gives rise to the name *box constraint* that is frequently used to refer to this formulation, since the vector α is constrained to lie inside the box with side length *C* in the positive orthant. The trade-off parameter between accuracy and regularisation directly controls the size of the α_i. This makes sense intuitively as the box constraints limit the influence of outliers, which would otherwise have large Lagrange multipliers. The constraint also ensures that the feasible region is bounded and hence that the primal always has a non-empty feasible region.

Remark 6.13

One problem with the soft margin approach suggested is the choice of parameter *C*. Typically a range of values must be tried before the best choice for a particular training set can be selected. Furthermore the scale of the parameter is affected by the choice of feature space. It has been shown, however, that the solutions obtained for different values of *C* in the optimisation problem (6.6) are the same as those obtained as ν is varied between 0 and 1 in the optimisation problem

$$\begin{array}{ll} \text{maximise} & W(\boldsymbol{\alpha}) = -\frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subj. to} & \sum_{i=1}^{\ell} y_i \alpha_i = 0, \\ & \sum_{i=1}^{\ell} \alpha_i \geq \nu \\ & 1/\ell \geq \alpha_i \geq 0, \; i = 1, \ldots, \ell. \end{array}$$

In this parametrisation ν places a lower bound on the sum of the α_i, which causes the linear term to be dropped from the objective function. It can be shown that the proportion of the training set that are margin errors is upper bounded by ν, while ν provides a lower bound on the total number of support vectors. Therefore ν gives a more transparent parametrisation of the problem which does not depend on the scaling of the feature space, but only on the noise level in the data. For details of this approach and its application to regression see pointers in Section 6.5.

In the case of the 1-norm margin slack vector optimisation the feasibility gap can be computed since the ξ_i are not specified when moving to the dual and so can be chosen to ensure that the primary problem is feasible, by taking

$$\xi_i = \max \left(0, 1 - y_i \left(\sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) + b \right) \right),$$

where is the current estimate for the dual problem and b has been chosen so that $y_i f(\mathbf{x}_i) = 1$ for some i with $C > {}_i > 0$. Once the primal problem is feasible the gap between the value of the primal and dual objectives becomes the sum of the Karush–Kuhn–Tucker complementarity conditions by the construction of the Lagrangian:

$$-L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r}) + \frac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle + C \sum_{i=1}^{\ell} \xi_i =$$

$$= \sum_{i=1}^{\ell} \alpha_i \left[y_i \left(\sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) + b \right) - 1 + \xi_i \right] + \sum_{i=1}^{\ell} r_i \xi_i,$$

where $r_i = C -$ ${}_i$. Hence, using the constraint on , the difference between primal and dual objectives is given by

$$\sum_{i=1}^{\ell} \alpha_i \left[y_i (\langle \mathbf{x}_i \cdot \mathbf{w} \rangle + b) - 1 + \xi_i \right] + \sum_{i=1}^{\ell} r_i \xi_i =$$

$$= \sum_{i=1}^{\ell} \alpha_i \left[y_i \left(\sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) \right) - 1 \right] + C \sum_{i=1}^{\ell} \xi_i$$

$$= \sum_{i,j=1}^{\ell} \alpha_i y_i y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) - \sum_{i=1}^{\ell} \alpha_i + C \sum_{i=1}^{\ell} \xi_i$$

$$= \sum_{i=1}^{\ell} \alpha_i - 2W(\boldsymbol{\alpha}) + C \sum_{i=1}^{\ell} \xi_i.$$

Remark 6.14

This explains why we noted that the maximal (or hard) margin case is an important concept in the solution of more sophisticated versions of the machine: both the 1- and the 2-norm soft margin machines lead to optimisation problems that are solved by relating them to the maximal margin case.

Remark 6.15

Historically, the soft margin machines were introduced before their justification in terms of margin distribution generalisation bounds. For this reason the 1-norm was preferred as it appeared closer to the percentile error bound. The results show that both 1- and 2-norm bounds on generalisation exist. The approach that performs better in practice will depend on the data and may be influenced by the type of noise that has influenced it.

Remark 6.16

The techniques developed for two-class classification have been generalised to the case where there are several categories. References to relevant papers will be given at the end of the chapter in Section 6.5.

Figure 6.3: This figure shows the decision boundaries that arise when using a Gaussian kernel with a fixed value of in the three different machines: (a) the maximal margin SVM, (b) the 2-norm soft margin SVM, and (c) the 1-norm soft margin SVM. The data are an artificially created two dimensional set, the white points being positive examples and the black points negative: the larger sized points are the support vectors. The red area comprises those points that are positively classified by the decision function, while the area classified negative is coloured blue. The size of the functional margin is indicated by the level of shading. Notice that the hard margin correctly classifies the whole training set at the expense of a more complex decision boundary. The two soft margin approaches both give smoother decision boundaries by misclassifying two positive examples

6.1.3 Linear Programming Support Vector Machines

Rather than using generalisation bounds based on margin distribution, one could try to enforce other learning biases, such as the sample compression bounds, as given in Theorems 4.25 and 6.8. This would lead for example to an algorithm for finding the sparsest separating hyperplane, regardless of its margin. The problem is computationally hard but can be approximated by minimising an estimate of the number of positive multipliers, $\sum_{i=1} {}_i$, while enforcing a margin of 1. Introducing slack variables in a way similar to that given above and working directly in the dual representation, one obtains the following linear optimisation problem:

$$\begin{array}{ll} \text{minimise} & L(\boldsymbol{\alpha}, \boldsymbol{\xi}) = \sum_{i=1}^{\ell} \alpha_i + C \sum_{i=1}^{\ell} \xi_i, \\ \text{subject to} & y_i \left[\sum_{j=1}^{\ell} \alpha_i \langle \mathbf{x}_i, \mathbf{x}_j \rangle + b \right] \geq 1 - \xi_i, \; i = 1, \ldots, \ell, \\ & \alpha_i \geq 0, \; \xi_i \geq 0, \; i = 1, \ldots, \ell. \end{array}$$

This type of approach was developed independently of the 2-norm maximal margin implicit in the definition of a standard Support Vector Machine. It has the advantage of relying on solving a linear programming problem as opposed to a convex quadratic one. The application of kernels to move to implicit feature spaces has also been made in this setting. Bounds on the generalisation directly in terms of $\sum_{i=1} {}_i$ have been derived more recently.

6.2 Support Vector Regression

The Support Vector method can also be applied to the case of regression, maintaining all the main features that characterise the maximal margin algorithm: a non-linear function is learned by a linear learning machine in a kernel-induced feature space while the capacity of the system is controlled by a parameter that does not depend on the dimensionality of the space. As in the classification case the learning algorithm minimises a convex functional and its solution is sparse.

As with the classification approach we motivate the approach by seeking to optimise the generalisation bounds given for regression in Chapter 4. These relied on defining a loss function that ignored errors that were within a certain distance of the true value. This type of function is referred to as an -insensitive loss function. Since this terminology is quite standard, we will risk using for this loss despite previously reserving this symbol for the generalisation error, that is the probability of misclassifying a randomly drawn test example.

Figure 6.4 shows an example of a one dimensional linear regression function with an -insensitive band. The variables measure the cost of the errors on the training points. These are zero for all points inside the band. Figure 6.5 shows a similar situation for a non-linear regression function.

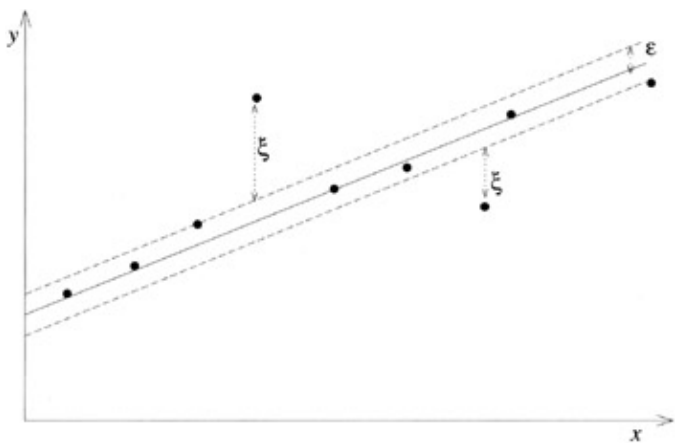


Figure 6.4: The insensitive band for a one dimensional linear regression problem

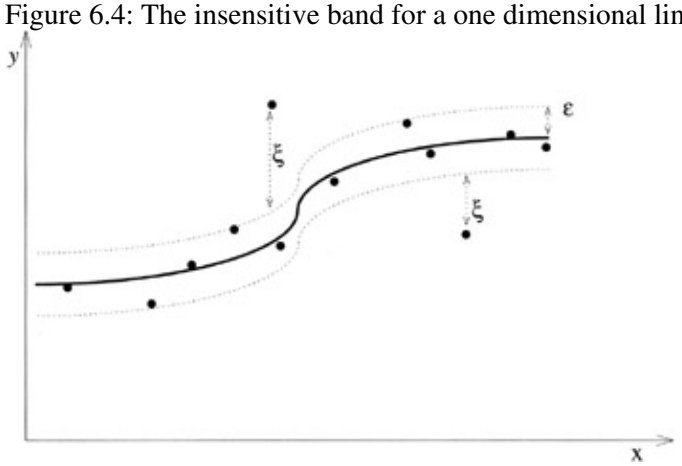


Figure 6.5: The insensitive band for a non-linear regression function

With many reasonable choices of loss function, the solution will be characterised as the minimum of a convex functional. Another motivation for considering the -insensitive loss function is that as with classification Support Vector Machines it will ensure sparseness of the dual variables. The idea of representing the solution by means of a small subset of training points has enormous computational advantages. Using the -insensitive loss function has that advantage, while still ensuring the existence of a global minimum and the optimisation of a reliable generalisation bound.

In this section we will first describe the -insensitive loss and then derive two algorithms from the bounds of Chapter 4, relating to the 1- or 2-norm of the loss vector. For comparison we will then give an algorithm for ridge regression in feature space, which does not enforce sparseness and hence presents more implementation problems. Finally, we will show how a popular regression algorithm based on Gaussian processes is equivalent to performing ridge regression in a feature space, and hence is intimately related to Support Vector Machines.

6.2.1 -Insensitive Loss Regression

Theorems 4.28 and 4.30 bound the generalisation performance of a linear regressor in terms of the norm of the weight vector and the 2- and 1-norms of the slack variables. The -insensitive loss function is equal to these slack variables.

Definition 6.17

The *(linear) -insensitive loss function L* (**x**, *y*, *f*) is defined by

$$L^{\varepsilon}(\mathbf{x}, y, f) = |y - f(\mathbf{x})|_{\varepsilon} = \max\left(0, |y - f(\mathbf{x})| - \varepsilon\right),$$

where *f* is a real-valued function on a domain *X*, **x** *X* and y $\mathbb{R}$. Similarly the *quadratic -insensitive loss* is given by

$$L_2^{\varepsilon}(\mathbf{x}, y, f) = |y - f(\mathbf{x})|_{\varepsilon}^2.$$

If we compare this loss function with the margin slack vector defined in Definition 4.27 it is immediate that the margin slack variable $((\mathbf{x}_i, y_i), f, \quad , \quad)$ satisfies

$$\xi\left((\mathbf{x}_i, y_i), f, \theta, \gamma\right) = L^{\theta-\gamma}(\mathbf{x}_i, y_i, f).$$

Hence as indicated above the results of Chapter 4 use an -insensitive loss function with = – . Figures 6.6 and 6.7 show the form of the linear and quadratic -insensitive losses for zero and non-zero as a function of $y - f(\mathbf{x})$.

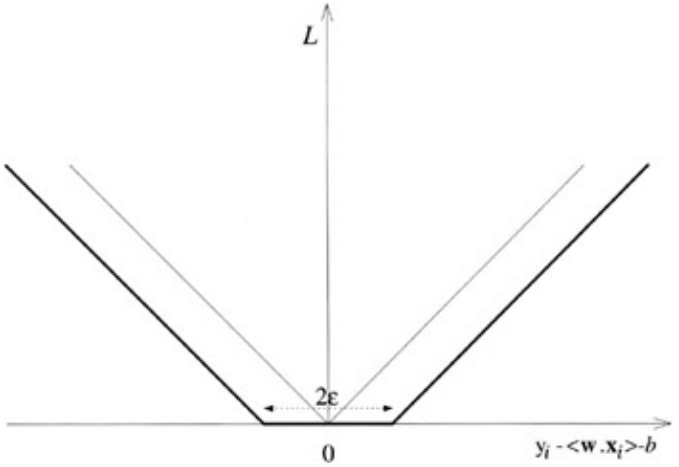


Figure 6.6: The linear -insensitive loss for zero and non-zero

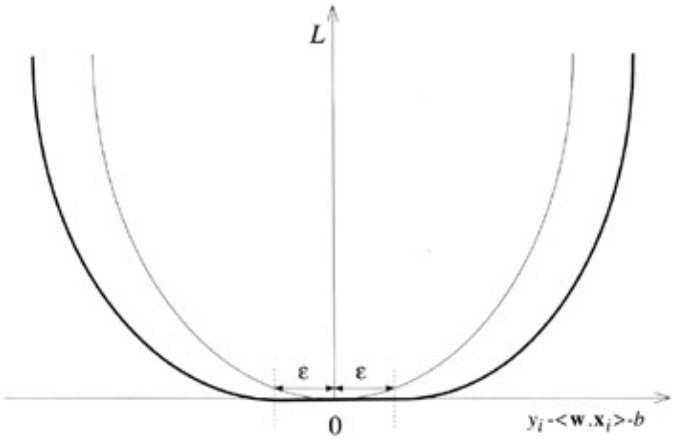


Figure 6.7: The quadratic -insensitive loss for zero and non-zero

Quadratic -Insensitive Loss

Theorem 4.28 suggests that we can optimise the generalisation of our regressor by minimising the sum of the quadratic -insensitive losses

$$R^2 \|\mathbf{w}\|^2 + \sum_{i=1}^{\ell} L_2^{\varepsilon}(\mathbf{x}_i, y_i, f),$$

where f is the function defined by the weight vector **w.** Minimising this quantity has the advantage of minimising the bound for all values of , which implies that it is minimised for all values of = + . As the classification case we introduce a parameter C to measure the trade-off between complexity and losses. The primal problem can therefore be defined as follows:

$$\begin{array}{ll} \text{minimise} & \|\mathbf{w}\|^2 + C \sum_{i=1}^{\ell} (\xi_i^2 + \hat{\xi}_i^2), \\ \text{subject to} & (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - y_i \leq \varepsilon + \xi_i, \; i = 1, \ldots, \ell, \\ & y_i - (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \leq \varepsilon + \hat{\xi}_i, \; i = 1, \ldots, \ell, \\ & \xi_i, \hat{\xi}_i \geq 0, \; i = 1, \ldots, \ell, \end{array} \qquad (6.7)$$

where we have introduced two slack variables, one for exceeding the target value by more than , and the other for being more than below the target. We will again typically consider solving this for a range of values of C and then use some validation method to select the optimal value of this parameter. The dual problem can be derived using the standard method and taking into account that $\xi_i \hat{\xi}_i = 0$ and therefore that the same relation $\alpha_i \hat{\alpha}_i = 0$ holds for the corresponding Lagrange multipliers:

$$\begin{array}{ll} \text{maximise} & \sum_{i=1}^{\ell} y_i(\hat{\alpha}_i - \alpha_i) - \varepsilon \sum_{i=1}^{\ell} (\hat{\alpha}_i + \alpha_i) \\ & \quad -\frac{1}{2} \sum_{i,j=1}^{\ell} (\hat{\alpha}_i - \alpha_i)(\hat{\alpha}_i - \alpha_j) \left(\langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle + \frac{1}{C} \delta_{ij} \right), \\ \text{subject to} & \sum_{i=1}^{\ell} (\hat{\alpha}_i - \alpha_i) = 0, \\ & \hat{\alpha}_i \geq 0, \; \alpha_i \geq 0, \; i = 1, \ldots, \ell. \end{array}$$

The corresponding Karush–Kuhn–Tucker complementarity conditions are

$$\begin{aligned} \alpha_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b - y_i - \varepsilon - \xi_i \right) &= 0, \quad i = 1, \ldots, \ell, \\ \hat{\alpha}_i \left(y_i - \langle \mathbf{w} \cdot \mathbf{x}_i \rangle - b - \varepsilon - \hat{\xi}_i \right) &= 0, \quad i = 1, \ldots, \ell, \\ \xi_i \hat{\xi}_i = 0, \; \alpha_i \hat{\alpha}_i &= 0, \quad i = 1, \ldots, \ell, \end{aligned}$$

Remark 6.18

Note that by substituting $\boldsymbol{\beta} = \hat{\boldsymbol{\alpha}} - \boldsymbol{\alpha}$ and using the relation $\alpha_i \hat{\alpha}_i = 0$, it is possible to rewrite the dual problem in a way that more closely resembles the classification case.

$$\begin{array}{ll} \text{maximise} & \sum_{i=1}^{\ell} y_i \beta_i - \varepsilon \sum_{i=1}^{\ell} |\beta_i| - \frac{1}{2} \sum_{i,j=1}^{\ell} \beta_i \beta_j \left(\langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle + \frac{1}{C} \delta_{ij} \right), \\ \text{subject to} & \sum_{i=1}^{\ell} \beta_i = 0, \; i = 1, \ldots, \ell. \end{array}$$

For $y \in \{-1, 1\}$, the similarity becomes even more apparent when $\varepsilon = 0$ and if we use the variables $\hat{\beta}_i = y_i\beta_i$, the only difference being that $\hat{\beta}_i$ is not constrained to be positive unlike the corresponding α_i in the classification case. We will in fact use α_i in place of **β** when we use this form later.

Remark 6.19

For non-zero ε the effect is to introduce an extra weight decay factor involving the dual parameters. The case $\varepsilon = 0$ corresponds to considering standard least squares linear regression with a weight decay factor controlled by the parameter C. As $C \to$ ∞, the problem tends to an unconstrained least squares, which is equivalent to leaving the inner product matrix diagonal unchanged. Note that references to investigations of more general loss functions will be given at the end of the chapter.

Hence, we have the following result in which we have moved directly to the more general kernel version.

Proposition 6.20

Suppose that we wish to perform regression on a training set

$$S = ((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_\ell, y_\ell)),$$

using the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$, and suppose the parameters α^* solve the following quadratic optimisation problem:

$$\begin{aligned} \textit{maximise} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} y_i\alpha_i - \varepsilon \sum_{i=1}^{\ell} |\alpha_i| \\ & \quad -\tfrac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i\alpha_j \left(K(\mathbf{x}_i, \mathbf{x}_j) + \frac{1}{C}\delta_{ij}\right), \\ \textit{subject to} \quad & \sum_{i=1}^{\ell} \alpha_i = 0. \end{aligned}$$

Let $f(\mathbf{x}) = \sum_{i=1} \alpha_i^* K(\mathbf{x}_i, \mathbf{x}) + b^*$, where b^* is chosen so that $f(\mathbf{x}_i) - y_i = -\varepsilon - \alpha_i^*/C$ for any i with $\alpha_i^* > 0$. Then the function $f(\mathbf{x})$ is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(\mathbf{x},\mathbf{z})$ that solves the optimisation problem (6.7).

Linear ε-Insensitive Loss

In Theorem 4.30 we must minimise the sum of the linear ε-insensitive losses

$$\frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{\ell} L^{\varepsilon}(\mathbf{x}_i, y_i, f),$$

for some value of the parameter C, which as in the classification case can be seen to control the size of $\|\mathbf{w}\|$ for a fixed training set. The equivalent primal optimisation problem is as follows.

$$\begin{aligned} \text{minimise} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 + C\textstyle\sum_{i=1}^{\ell}(\xi_i + \hat{\xi}_i), \\ \text{subject to} \quad & (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - y_i \le \varepsilon + \xi_i, \\ & y_i - (\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) \le \varepsilon + \hat{\xi}_i, \\ & \xi_i, \hat{\xi}_i \ge 0, \ i = 1, 2, \dots, \ell. \end{aligned} \qquad (6.8)$$

The corresponding dual problem can be derived using the now standard techniques:

$$\begin{aligned} \text{maximise} \quad & \textstyle\sum_{i=1}^{\ell}(\hat{\alpha}_i - \alpha_i)y_i - \varepsilon\sum_{i=1}^{\ell}(\hat{\alpha}_i + \alpha_i) \\ & \quad -\tfrac{1}{2}\textstyle\sum_{i,j=1}^{\ell}(\hat{\alpha}_i - \alpha_i)(\hat{\alpha}_j - \alpha_j)\langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle, \\ \text{subject to} \quad & 0 \le \alpha_i, \hat{\alpha}_i \le C, \ i = 1, \dots, \ell, \\ & \textstyle\sum_{i=1}^{\ell}(\hat{\alpha}_i - \alpha_i) = 0, \ i = 1, \dots, \ell. \end{aligned}$$

The corresponding Karush–Kuhn–Tucker complementarity conditions are

$$\alpha_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b - y_i - \varepsilon - \xi_i \right) = 0, \quad i = 1, \ldots, \ell,$$
$$\hat{\alpha}_i \left(y_i - \langle \mathbf{w} \cdot \mathbf{x}_i \rangle - b - \varepsilon - \hat{\xi}_i \right) = 0, \quad i = 1, \ldots, \ell,$$
$$\xi_i \hat{\xi}_i = 0, \ \alpha_i \hat{\alpha}_i = 0, \quad i = 1, \ldots, \ell,$$
$$(\alpha_i - C)\, \xi_i = 0, \ (\hat{\alpha}_i - C)\, \hat{\xi}_i = 0, \quad i = 1, \ldots, \ell.$$

Again as mentioned in Remark 6.18 substituting $_i$ for $\hat{\alpha}_i - \alpha_i$, and taking into account that $\alpha_i \hat{\alpha}_i = 0$, we obtain the following proposition.

Proposition 6.21

Suppose that we wish to perform regression on a training sample

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)),$$

using the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$, and suppose the parameters * solve the following quadratic optimisation problem:

maximise $W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} y_i \alpha_i - \varepsilon \sum_{i=1}^{\ell} |\alpha_i| - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$,
subject to $\sum_{i=1}^{\ell} \alpha_i = 0, \ -C \le \alpha_i \le C, \ i = 1, \ldots, \ell.$

Let $f(\mathbf{x}) = \sum_{i=1}$ $_i^* K(\mathbf{x}_i, \mathbf{x}) + b^*$, where b^* is chosen so that $f(\mathbf{x}_i) - y_i = -$ for any i with $0 <$ $_i^* < C$. Then the function $f(\mathbf{x})$ is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$ that solves the optimisation problem (6.8).

Remark 6.22

If we consider the band of ± around the function output by the learning algorithm, the points that are not strictly inside the tube are support vectors. Those not touching the tube will have the absolute value of that parameter equal to C.

Remark 6.23

We have again described the most standard optimisations considered. A number of variations have been considered in the literature including considering different norms as well as adapting the optimisation to control the number of points lying outside the -band. In this case the number of points is given as an input to the problem rather than the value of . References to this and other developments in the use of SVMs for regression will be given at the end of the chapter in Section 6.5.

6.2.2 Kernel Ridge Regression

As mention in Remark 6.19 the case = 0 for the quadratic loss corresponds to least squares regression with a weight decay factor. This approach to regression is also known as ridge regression, and we will see shortly that it is equivalent to the techniques derived from Gaussian processes. So we will give it an independent derivation, which highlights the connections with those systems. These systems also ignore the bias term. The (primal) problem can therefore be stated as follows:

$$\begin{aligned} &\text{minimise} && \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^{\ell} \xi_i^2, \\ &\text{subject to} && y_i - \langle \mathbf{w} \cdot \mathbf{x}_i \rangle = \xi_i, \ i = 1, \ldots, \ell, \end{aligned} \tag{6.9}$$

from which we derive the following Lagrangian

$$\text{minimise} \quad L(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^{\ell} \xi_i^2 + \sum_{i=1}^{\ell} \alpha_i (y_i - \langle \mathbf{w} \cdot \mathbf{x}_i \rangle - \xi_i).$$

Differentiating and imposing stationarity, we obtain that

$$\mathbf{w} = \frac{1}{2\lambda}\sum_{i=1}^{\ell}\alpha_i\mathbf{x}_i \text{ and } \xi_i = \frac{\alpha_i}{2}.$$

Resubstituting these relations gives the following dual problem:

$$\text{maximise} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} y_i\alpha_i - \frac{1}{4\lambda}\sum_{i,j=1}^{\ell}\alpha_i\alpha_j\langle \mathbf{x}_i \cdot \mathbf{x}_j\rangle - \frac{1}{4}\sum\alpha_i^2,$$

that for convenience we rewrite in vector form:

$$W(\boldsymbol{\alpha}) = \mathbf{y}'\boldsymbol{\alpha} - \frac{1}{4\lambda}\boldsymbol{\alpha}'\mathbf{K}\boldsymbol{\alpha} - \frac{1}{4}\boldsymbol{\alpha}'\boldsymbol{\alpha},$$

where $\mathbf{K}$ denotes the Gram matrix $\mathbf{K}_{ij} = \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$, or the kernel matrix $\mathbf{K}_{ij} K(\mathbf{x}_i, \mathbf{x}_j)$, if we are working in a kernel-induced feature space. Differentiating with respect to and imposing stationarity we obtain the condition

$$-\frac{1}{2\lambda}\mathbf{K}\boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha} + \mathbf{y} = 0,$$

giving the solution

$$\boldsymbol{\alpha} = 2\lambda(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y}$$

and the corresponding regression function

$$f(\mathbf{x}) = \langle \mathbf{w} \cdot \mathbf{x}\rangle = \mathbf{y}'(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{k}$$

where $\mathbf{k}$ is the vector with entries $k_i = \langle \mathbf{x}_i \cdot \mathbf{x} \rangle$, $i = 1,\ldots,$. Hence, we have the following proposition.

Proposition 6.24

Suppose that we wish to perform regression on a training sample

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)),$$

using the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$, and let $f(\mathbf{x}) = \mathbf{y}$ $(\mathbf{K} +$ $\mathbf{I})^{-1}\mathbf{k}$, where $\mathbf{K}$ is the $\times$ matrix with entries $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ and $\mathbf{k}$ is the vector with entries $\mathbf{k}_i = K(\mathbf{x}_i, \mathbf{x})$. Then the function $f(\mathbf{x})$ is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$ that solves the ridge regression optimisation problem (6.9).

This algorithm has appeared independently under a number of different names. It is also known as Krieging and the solutions are known as regularisation networks, where the regulariser has been implicitly selected by the choice of kernel. We will see in the next subsection that the same function results when we solve the Bayesian learning problem using Gaussian processes.

6.2.3 Gaussian Processes

This subsection will bring together the discussion of Bayesian learning from Section 4.6 with the idea of a Gaussian process introduced in Section 3.5. The posterior distribution is given by

$$P(t, \mathbf{t}|\mathbf{x}, S) \propto P(\mathbf{y}|\mathbf{t})P(t, \mathbf{t}|\mathbf{x}, \mathbf{X}),$$

$\mathbf{y}$ are the output values from the training set, which are assumed to be corrupted by noise and $\mathbf{t}$ are the true target output values related to $\mathbf{y}$ by the distribution,

$$P(\mathbf{y}|\mathbf{t}) \propto \exp\left[-\frac{1}{2}(\mathbf{y} - \mathbf{t})'\Omega^{-1}(\mathbf{y} - \mathbf{t})\right],$$

where Ω = $^2\mathbf{I}$. The Gaussian process distribution was introduced in Section 3.5 and is defined as

$$P(t, \mathbf{t}|\mathbf{x}, \mathbf{X}) = P_{f\sim\mathcal{D}}\left[(f(\mathbf{x}), f(\mathbf{x}_1), \ldots, f(\mathbf{x}_\ell)) = (t, t_1, \ldots, t_\ell)\right]$$
$$\propto \exp\left(-\frac{1}{2}\hat{\mathbf{t}}'\hat{\Sigma}^{-1}\hat{\mathbf{t}}\right),$$

where $\hat{\mathbf{t}} = (t, t_1, \ldots, t_\ell)'$ and $\hat{\Sigma}$ is indexed with rows and columns from 0 to . The principal submatrix on the rows 1 to , is the matrix Σ, where Σ$_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ for the covariance function $K(\mathbf{x}, \mathbf{z})$, while the entry $\hat{\Sigma}_{00} = K(\mathbf{x}, \mathbf{x})$ and the entries in the 0 row and column are

$$\hat{\Sigma}_{0i} = \hat{\Sigma}_{i0} = K(\mathbf{x}, \mathbf{x}_i).$$

The distribution of the variable t is the predictive distribution. It is a Gaussian distribution with mean $f(\mathbf{x})$ and variance $V(\mathbf{x})$, given by

$$f(\mathbf{x}) = \mathbf{y}'(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k},$$
$$V(\mathbf{x}) = K(\mathbf{x}, \mathbf{x}) - \mathbf{k}'(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}, \qquad (6.10)$$

where $\mathbf{K}$ is the × matrix with entries $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ and $\mathbf{k}$ is the vector with entries $\mathbf{k}_i = K(\mathbf{x}_i, \mathbf{x})$. Hence, the prediction made by the Gaussian process estimate coincides exactly with the ridge regression function of Proposition 6.24, where the parameter has been chosen equal to the variance of the noise distribution. This reinforces the relationship between margin slack optimisation and noise in the data. It suggests that optimising the 2-norm (see problem (6.7)) corresponds to an assumption of Gaussian noise, with variance equal to $\frac{1}{C}$.

The Gaussian process also delivers an estimate for the reliability of the prediction in the form of the variance of the predictive distribution. More importantly the analysis can be used to estimate the evidence in favour of a particular choice of covariance function. This can be used to adaptively choose a parameterised kernel function which maximises the evidence and hence is most likely given the data. The covariance of kernel function can be seen as a model of the data and so this provides a principled method for model selection.

6.3 Discussion

This chapter contains the core material of the book. It shows how the learning theory results of Chapter 4 can be used to avoid the difficulties of using linear functions in the high dimensional kernel-induced feature spaces of Chapter 3. We have shown how the optimisation problems resulting from such an approach can be transformed into dual convex quadratic programmes for each of the approaches adopted for both classification and regression. In the regression case the loss function used only penalises errors greater than a threshold . Such a loss function typically leads to a sparse representation of the decision rule giving significant algorithmic and representational advantages. If, however, we set = 0 in the case of optimising the 2-norm of the margin slack vector, we recover the regressor output by a Gaussian process with corresponding covariance function, or equivalently the ridge regression function. These approaches have the disadvantage that since = 0, the sparseness of the representation has been lost.

The type of criterion that is optimised in all of the algorithms we have considered also arises in many other contexts, which all lead to a solution with a dual representation. We can express these criteria in the general form

$$\|f\|_{\mathscr{H}}^2 + C\frac{1}{\ell}\sum_{i=1}^{\ell} L(y_i, f(\mathbf{x}_i)),$$

where L is a loss function, $\|\cdot\|$ a regulariser and C is the regularisation parameter. If L is the square loss, this gives rise to regularisation networks of which Gaussian processes are a special case. For this type of problem the solution can always be expressed in the dual form.

In the next chapter we will describe how these optimisation problems can be solved efficiently, frequently making use of the sparseness of the solution when deriving algorithms for very large datasets.

6.4 Exercises

1. What is the relation between the four expressions $W(\ )$, $\sum_{i=1} \quad _i$,

$$\sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j),$$

and $\frac{1}{\gamma^2}$, when is the solution of the optimisation problem (6.2) and is the corresponding geometric margin? What happens to the optimisation problem if the data are not linearly separable? How is this problem avoided for the soft margin optimisation problems?
2. Derive the dual optimisation problem of the regression problem (6.7), hence demonstrating Proposition 6.20.
3. Consider the optimisation problem

$$\begin{array}{ll} \text{minimise}_{\mathbf{w},b} & \langle \mathbf{w} \cdot \mathbf{w} \rangle + C_1 \sum_{i=1}^{\ell} \xi_i + C_2 \sum_{i=1}^{\ell} \xi_i^2, \\ \text{subject to} & y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) \geq 1 - \xi_i, \ i = 1, \ldots, \ell, \\ & \xi_i \geq 0, \ i = 1, \ldots, \ell. \end{array}$$

Discuss the effect of varying the parameters C_1 and C_2. Derive the dual optimisation problem.

6.5 Further Reading and Advanced Topics

Support Vector Machines are a very specific class of algorithms, characterised by the use of kernels, the absence of local minima, the sparseness of the solution and the capacity control obtained by acting on the margin, or on other ‘dimension independent’ quantities such as the number of support vectors. They were invented by Boser, Guyon and Vapnik [19], and first introduced at the Computational Learning Theory (COLT) 1992 conference with the paper [19]. All of these features, however, were already present and had been used in machine learning since the 1960s: large margin hyperplanes in the *input* space were discussed for example by Duda and Hart [35], Cover [28], Vapnik et al. [166] [161], and several statistical mechanics papers (for example [4]); the use of kernels was proposed by Aronszajn [7], Wahba [171], Poggio [116], and others, but it was the paper by Aizermann et al. [1] in 1964 that introduced the geometrical interpretation of the kernels as inner products in a feature space. Similar optimisation techniques were used in pattern recognition by Mangasarian [84], and the sparseness had also already been discussed [28]. See also [57] for related early work. The use of slack variables to overcome the problem of noise and non-separability was also introduced in the 1960s by Smith [143] and improved by Bennett and Mangasarian [15]. However, it was not until 1992 that all of these features were put together to form the maximal margin classifier, the basic Support Vector Machine, and not until 1995 that the soft margin version [27] was introduced: it is surprising how naturally and elegantly all the pieces fit together and complement each other. The papers [138], [10] gave the first rigorous statistical bound on the generalisation of hard margin SVMs, while the paper [141] gives similar bounds for the soft margin algorithms and for the regression case.

After their introduction, an increasing number of researchers have worked on both the algorithmic and theoretical analysis of these systems, creating in just a few years what is effectively a new research direction in its own right, merging concepts from disciplines as distant as statistics, functional analysis, optimisation, as well as machine learning. The soft margin classifier was introduced a few years later by Cortes and Vapnik [27], and in 1995 the algorithm was extended to the regression case [158].

The two recent books written by Vapnik [158, 159] provide a very extensive theoretical background of the field and develop the concept of a Support Vector Machine.

Since most recent advances in kernels, learning theory, and implementation are discussed at the ends of the relevant chapters, we only give a brief overview of some of the improvements recently obtained from the point of view of the overall algorithm. Work has been done on generalisations of the method [88], and extensions to the multi-class case [178], [159], [113]. More general regression scenarios have also been studied: Smola, Schölkopf and Mueller discussed a very general class of loss functions [147], and the use of ridge regression in feature space was considered by [144] and [125].

Ridge regression is a special case of regularisation networks. The concept of regularisation was introduced by Tikhonov [153], and applied to learning in the form of regularisation networks by Girosi et al. [52]. The relation between regularisation networks and Support Vector Machines has been explored by a number of authors [51], [171], [172], [146], [38]. The connection between regularisation networks and neural networks was explored as early as 1990 in [116], see also [52] and [39] for a full bibliography.

The ν-Support Vector algorithm for classification and regression described in Remark 6.13 was introduced in [135] and further developed in [131] and [130]. Other adaptations of the basic approach have been used for density estimation [167], transduction [13], Bayes point estimation [59], ordinal regression [60], etc. Rifkin et al. [121] show that for some degenerate training sets the soft margin gives a trivial solution.

Theoretical advances that do not fit within the framework of Chapter 4 include the analyses of generalisation given in the article [34], which provides a statistical mechanical analysis of SVMs, the papers [66], [160], [177], [173], [107], which provide a cross-validation analysis of the expected error, and the book [159], which gives an expected error bound in terms of the margin and radius of the smallest ball containing the essential support vectors.

Extensions of the SVM concept have been made by several authors, for example Mangasarian's generalised SVMs [83]. Particularly interesting is the development of a system, called the Bayes point machine [59], that enforces another inductive principle, given by Bayesian generalisation theory. Albeit losing the feature of sparseness, this system exhibits excellent performance, and illustrates the important point that this class of algorithms is not limited to the use of margin bounds. Another example of a system similar to SVMs that does not enforce a margin bound is given by Gaussian processes.

More recently, a number of practical applications of SVMs have been reported, in fields as diverse as bioinformatics, computational linguistics and computer vision (some of which are reported in Chapter 8). Many of these recent advances are reported in the collections [132], and [149], and in the surveys [23], [145], [39]. Most of the new contributions are only available from the internet, and can be accessed via the website [30].

Finally, the PhD dissertations of Cortes [26], Schölkopf [129] and Smola [148], provide a valuable first hand source of research topics including work on the feasibility gap.

Gaussian processes are surveyed in [180], and in the dissertation of Rasmussen [120]. Extensions of Gaussian processes to the classification case have also been undertaken but fall beyond the scope of this book.

These references are also given on the website **http://www.support-vector.net**, which will be kept up to date with new work, pointers to software and papers that are available on-line.

Chapter 7: Implementation Techniques

Overview

In the previous chapter we showed how the training of a Support Vector Machine can be reduced to maximising a convex quadratic form subject to linear constraints. Such convex quadratic programmes have no local maxima, and their solution can always be found efficiently. Furthermore this dual representation of the problem showed how the training could be successfully effected even in very high dimensional feature spaces. The problem of minimising differentiable functions of many variables has been widely studied, especially in the convex case, and most of the standard approaches can be directly applied to SVM training. However, in many cases specific techniques have been developed to exploit particular features of this problem. For example, the large size of the training sets typically used in applications is a formidable obstacle to a direct use of standard techniques, since just storing the kernel matrix requires a memory space that grows quadratically with the sample size, and hence exceeds hundreds of megabytes even when the sample size is just a few thousand points.

Such considerations have driven the design of specific algorithms for Support Vector Machines that can exploit the sparseness of the solution, the convexity of the optimisation problem, and the implicit mapping into feature space. All of these features help to create remarkable computational efficiency. The elegant mathematical characterisation of the solutions can be further exploited to provide stopping criteria and decomposition procedures for very large datasets.

In this chapter we will briefly review some of the most common approaches before describing in detail one particular algorithm, Sequential Minimal Optimisation (SMO), that has the additional advantage of not only being one of the most competitive but also being simple to implement. As an exhaustive discussion of optimisation algorithms is not possible here, a number of pointers to relevant literature and on-line software is provided in Section 7.8.

7.1 General Issues

The optimisation problem associated to classification Support Vector Machines can be written as follows:

$$\begin{array}{ll} \text{maximise} & W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j), \\ \text{subject to} & \sum_{i=1}^{\ell} \alpha_i y_i = 0, \\ & 0 \leq \alpha_i \leq C, \, i = 1, \ldots, \ell, \end{array} \tag{7.1}$$

where C = ∞ gives the hard margin case and with an adaptation of the kernel function the 2-norm soft margin optimisation, while C < ∞ gives the 1-norm soft margin case. In the regression case, the problem for the linear -insensitive loss is

$$\begin{array}{ll} \text{maximise} & W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} y_i \alpha_i - \varepsilon \sum_{i=1}^{\ell} |\alpha_i| - \frac{1}{2} \sum_{i,j=1}^{\ell} \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \\ \text{subject to} & \sum_{i=1}^{\ell} \alpha_i = 0, \, -C \leq \alpha_i \leq C, \, i = 1, \ldots, \ell, \end{array} \tag{7.2}$$

where setting C = ∞ and adding a constant to the diagonal of the kernel matrix delivers the quadratic -insensitive loss optimisation.

The convexity of the functional $W(\ )$ and of the feasible region ensure that the solution can always be found efficiently. The solution satisfies the Karush–Kuhn–Tucker complementarity conditions. For the classification maximal margin case they are

$$\alpha_i \left[y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) - 1 \right] = 0, \quad i = 1, \ldots, \ell.$$

For the 2-norm soft margin optimisation they are

$$\alpha_i \left[y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) - 1 + \xi_i \right] = 0, \quad i = 1, \ldots, \ell,$$

while for the 1-norm soft margin optimisation

$$\begin{array}{ll} \alpha_i \left[y_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \right) - 1 + \xi_i \right] = 0, & i = 1, \ldots, \ell, \\ (\alpha_i - C) \, \xi_i = 0, & i = 1, \ldots, \ell. \end{array}$$

In the regression case for the quadratic -insensitive loss function they are

$$\begin{array}{ll} \alpha_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b - y_i - \varepsilon - \xi_i \right) = 0, & i = 1, \ldots, \ell, \\ \hat{\alpha}_i \left(y_i - \langle \mathbf{w} \cdot \mathbf{x}_i \rangle - b - \varepsilon - \hat{\xi}_i \right) = 0, & i = 1, \ldots, \ell, \\ \xi_i \hat{\xi}_i = 0, \, \alpha_i \hat{\alpha}_i = 0, & i = 1, \ldots, \ell, \end{array}$$

and for the linear -insensitive loss function they are

$$\begin{array}{ll} \alpha_i \left(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b - y_i - \varepsilon - \xi_i \right) = 0, & i = 1, \ldots, \ell, \\ \hat{\alpha}_i \left(y_i - \langle \mathbf{w} \cdot \mathbf{x}_i \rangle - b - \varepsilon - \hat{\xi}_i \right) = 0, & i = 1, \ldots, \ell, \\ \xi_i \hat{\xi}_i = 0, \, \alpha_i \hat{\alpha}_i = 0, & i = 1, \ldots, \ell, \\ (\alpha_i - C) \, \xi_i = 0, \, (\hat{\alpha}_i - C) \, \hat{\xi}_i = 0, & i = 1, \ldots, \ell, \end{array}$$

As discussed below, in practice such conditions will only be approximated to a certain tolerance level. Most numerical strategies follow the approach of starting from an arbitrary feasible point and iteratively increasing the value of the dual objective function without leaving the feasible region, until a stopping criterion is satisfied. Additionally, some approaches implement heuristics that achieve this by acting only on a small subset of the $_i$ at each time, in order to improve the computational efficiency. Such techniques, which somehow exploit the sparseness of the problem, make it possible to scale to very large datasets (tens or hundreds of thousands of examples).

Stopping criteria can be obtained by exploiting the properties of convex optimisation problems in different ways. Based on the fact that the feasibility gap vanishes at the solution, one can monitor convergence by checking this quantity. Alternatively, one can stop when the increase in the dual objective function is less than a certain pre-fixed threshold. Finally, the Karush–Kuhn–Tucker conditions can be explicitly calculated and monitored in order to determine whether the solution has been found. We describe these three stopping criteria in more detail.

1. Monitoring the growth of the dual objective function. The quadratic dual objective function for the particular SVM optimisation problem achieves its maximum at the solution. Monitoring the value of the function, and especially the increase in that value at each step, provides the simplest stopping criterion. The training can be stopped when the fractional rate of increase of the objective function $W(\alpha)$ falls below a given tolerance (e.g. 10^{-9}). Unfortunately, this criterion has been shown to be unreliable and in some cases can deliver poor results.
2. Monitoring the Karush–Kuhn–Tucker conditions for the primal problem. They are necessary and sufficient conditions for convergence, so they provide the natural criterion. For example in the classification case the following criteria must be checked for the 1-norm soft margin optimisation:

$$0 \leq \alpha_i \leq C,$$

$$y_i f(\mathbf{x}_i) \begin{cases} \geq 1 & \text{for points with } \alpha_i = 0, \\ = 1 & \text{for points with } 0 < \alpha_i < C, \\ \leq 1 & \text{for points with } \alpha_i = C. \end{cases}$$

Notice that one does not need to compute the slack variables ξ_i for this case. For the 2-norm soft margin optimisation the criteria are:

$$\alpha_i \geq 0,$$

$$y_i f(\mathbf{x}_i) \begin{cases} \geq 1 & \text{for points with } \alpha_i = 0, \\ = 1 - \alpha_i/C & \text{for points with } \alpha_i > 0. \end{cases}$$

In this case the slack variable is implicitly defined by $\xi_i = \alpha_i/C$. Naturally these criteria must again be verified to within some chosen tolerance, for example a good choice in this case is to within 10^{-2}.

3. Another way to characterise the solution is by means of the gap between the primal and dual objective functions, since this vanishes only at the optimal point. We call this difference the feasibility gap to distinguish it from the duality gap of an optimisation problem, which is the difference between the values of the primal and dual solutions. For convex quadratic optimisation problems this difference is zero. In the case of the 1-norm soft margin optimisation the feasibility gap can be computed as described in Subsection 6.1.2. We first set

$$\xi_i = \max\left(0, 1 - y_i\left(\sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) + b\right)\right),$$

where α is the current estimate for the dual problem and b has been chosen so that $y_i f(\mathbf{x}_i) = 1$ for some i with $C > \alpha_i > 0$. The difference between primal and dual objectives is then given by

$$\sum_{i=1}^{\ell} \alpha_i \left[y_i\left(\sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i)\right) - 1\right] + C\sum_{i=1}^{\ell} \xi_i = \\ = \sum_{i=1}^{\ell} \alpha_i - 2W(\boldsymbol{\alpha}) + C\sum_{i=1}^{\ell} \xi_i,$$

where $W(\alpha)$ is the dual objective. The ratio

$$\frac{\text{primal obj.} - \text{dual obj.}}{\text{primal objective} + 1} = \frac{\sum_{i=1}^{\ell} \alpha_i - 2W(\boldsymbol{\alpha}) + C\sum_{i=1}^{\ell} \xi_i}{W(\boldsymbol{\alpha}) + \sum_{i=1}^{\ell} \alpha_i - 2W(\boldsymbol{\alpha}) + C\sum_{i=1}^{\ell} \xi_i + 1} = \frac{\sum_{i=1}^{\ell} \alpha_i - 2W(\boldsymbol{\alpha}) + C\sum_{i=1}^{\ell} \xi_i}{\sum_{i=1}^{\ell} \alpha_i - W(\boldsymbol{\alpha}) + C\sum_{i=1}^{\ell} \xi_i + 1} \quad (7.3)$$

provides a useful measure of progress, and checking if this is less than say 10^{-3} can be used as a stopping criterion.

For the maximal margin and 2-norm margin slack optimisation (implemented by adding a constant to the diagonal of the kernel matrix) we can imagine introducing a box constraint equal to the largest $_i$ at each iteration t, $C_t = \max_i (\ _i^t) + 1$. This will mean that the actual running of the algorithm will not be affected, but at any iteration the current can be regarded as a feasible solution of the corresponding box constraint optimisation. For this problem we can compute the feasibility gap using the above equations and for large enough $C > \max_i (\ _i$∗), where **∗** is the optimal solution of the unconstrained problem, the two solutions coincide.

Remark 7.1

Note that the tolerance level used to verify the stopping criteria is very important. Achieving high levels of accuracy can be very time consuming but may not give any significant advantage in prediction accuracy. So in practice one needs to set tolerance levels to ensure approximately optimal conditions.

Remark 7.2

An important consequence of the stopping criteria is that they can also motivate heuristics to speed convergence: for example one can expect to approach the solution faster by acting on points that contribute more to the feasibility gap, for example those for which

$$\alpha_i \left[y_i \left(\sum_{j=1}^{\ell} y_j \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) \right) - 1 \right] + C\xi_i$$

is large, or by performing operations that lead to a larger increase in the dual objective function. These considerations will be exploited in the following sections to design more effective training algorithms.

Remark 7.3

It should be noted that even if the solution **w** is unique, its expansion in terms of may not be if the kernel matrix is only positive semi-definite.

In the rest of the chapter, we will first discuss a simple gradient ascent algorithm that does not optimise the bias. This will provide a vehicle for introducing a series of important concepts that will be needed for the more sophisticated approaches in the later sections.

4 Chapter 7: Implementation Techniques

7.2 The Naive Solution: Gradient Ascent

The simplest numerical solution of a convex optimisation problem is obtained by gradient ascent, sometimes known as the steepest ascent algorithm. The algorithm starts with an initial estimate for the solution, denoted by ^{0}, and then iteratively updates the vector following the steepest ascent path, that is moving in the direction of the gradient of $W(\ )$ evaluated at the position ^{t} for update $t + 1$. At each iteration the direction of the update is determined by the steepest ascent strategy but the length of the step still has to be fixed. The length of the update is known as the *learning rate*.

In the sequential or *stochastic* version, this strategy is approximated by evaluating the gradient for just one pattern at a time, and hence updating a single component $_i^t$ by the increment

$$\delta\alpha_i^t = \eta \frac{\partial W(\boldsymbol{\alpha}^t)}{\partial \alpha_i}$$

where the parameter is the learning rate. If is chosen carefully, the objective function will increase monotonically, and the average direction approximates the local gradient. One can modify as a function of time, or as a function of the input pattern being learned, in order to improve the convergence. The choice of is a delicate one; too large values can cause the system to oscillate without converging to the solution, while too small values result in very slow rates of convergence. Following this strategy means that the direction of each step is parallel to one of the axes and so is known a priori. The algorithm determines the step length and more importantly its sign. One further freedom available is the choice of the ordering in which the points are updated, as some points may cause a greater movement along the path towards a solution. This will be discussed further below.

Another way of looking at the same algorithm is that the quantity $W(\ )$ is iteratively increased by freezing all variables but one. Hence a multi-dimensional problem is reduced to a sequence of one dimensional ones. The uniqueness of the global maximum guarantees that for suitable choices of the algorithm will always find the solution. Such a strategy is usually not optimal from the point of view of speed, but is surprisingly good for datasets of up to a couple of thousand points and has the advantage that its implementation is very straightforward. As we will see later, another major advantage is that it only acts on one training point at a time, and so does not require that we store all the data simultaneously in fast memory.

Using this approach, one source of problems is the linear constraint

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0 \qquad (7.4)$$

derived from optimising the bias b in the decision function. This constraint defines a hyperplane in the dimensional space of parameters, and restricts the feasible region to the intersection between this hyperplane and the hypercube of side C for the 1-norm soft margin optimisation or the positive orthant in the case of the maximal margin and 2-norm soft margin optimisation. A natural strategy for enforcing such a constraint is to make sure that the current solution never leaves the feasible region. Unfortunately, updating one component at a time makes this impossible: if at the time t the constraint of equation (7.4) is satisfied, then after performing a non-trivial update on one $_i$ it will cease to hold. The minimum number of multipliers that can be simultaneously updated without leaving the feasible region is hence 2, and this consideration will form the basis of the SMO algorithm, described later, in Section 7.5. Other strategies also exist, such as for example enforcing the constraint

$$-\zeta_t \leq \sum_{i=1}^{\ell} \alpha_i^t y_i \leq \zeta_t,$$

where $_t$ is decreased at each iteration. In this section we will discuss the algorithm for the case where the bias b is fixed a priori and hence the equality constraint does not need to be enforced.

Remark 7.4

Fixing the bias beforehand may seem restrictive, but if we consider the embedding of the input space X into a space $\hat{X}$ of one extra dimension, in which we denote the new vector by $\hat{\mathbf{x}} = (\mathbf{x}, \tau)$, for some fixed value τ, then the linear function on X represented by a unit weight vector $\mathbf{w}$ and bias b is equivalent to the function with weight vector $\hat{\mathbf{w}} = (\mathbf{w}, b/\tau)$ and zero bias in the space $\hat{X}$, since

$$\langle \mathbf{w} \cdot \mathbf{x} \rangle + b = \langle \hat{\mathbf{w}} \cdot \hat{\mathbf{x}} \rangle .$$

Note that for a non-trivial classification of a training set

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell))$$

we must have $b \leq \boldsymbol{R} = \max_{1 \leq i \leq} (\|\mathbf{x}_i\|)$. Adding this extra dimension to a feature space induced by the kernel K is equivalent to adding τ^2 to create the adapted kernel

$$\hat{K}(\mathbf{x}, \mathbf{z}) = K(\mathbf{x}, \mathbf{z}) + \tau^2.$$

The only drawback of taking this route is that the geometric margin of the separating hyperplane in the augmented space will typically be less than that in the original space. For poor choices of τ this difference can be very large, affecting both the convergence of the algorithm and generalisation performance of the resulting classifier. Assuming that $\mathbf{w}$ is normalised, the functional margin of the new weight vector $\hat{\mathbf{w}}$ on the set $\hat{S}$ is equal to the geometric margin of $\mathbf{w}$ on S. Hence the geometric margin of $\hat{\mathbf{w}}$ is

$$\begin{aligned} \gamma \, \|\hat{\mathbf{w}}\|^{-1} &= \gamma \left(1 + b^2/\tau^2\right)^{-1/2} \\ &\leq \gamma \left(1 + R^2/\tau^2\right)^{-1/2}. \end{aligned}$$

The quantity that measures the fat-shattering dimension in the space X (see Theorem 4.16 in Chapter 4) is the ratio

$$\frac{\max_{1 \leq i \leq \ell} \left(\|\mathbf{x}_i\|^2 \right)}{\gamma^2} = \frac{R^2}{\gamma^2},$$

while in $\hat{X}$ this ratio becomes

$$\frac{\max_{1 \leq i \leq \ell} \left(\|\mathbf{x}_i\|^2 \right) + \tau^2}{\gamma^2 \left(1 + R^2/\tau^2\right)^{-1}} = \frac{\left(R^2 + \tau^2\right)\left(1 + R^2/\tau^2\right)}{\gamma^2}.$$

The right hand side of the inequality is minimised by taking τ = R, when it becomes $4R^2/$ ^{2}. Hence, a safe choice of τ is R, in that it will only increase the bound on the fat-shattering dimension by a factor of 4.

If we set the bias to a fixed value then the dual of the optimisation problem becomes

$$\begin{aligned} &\text{maximise} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{\ell} \alpha_i - \tfrac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j), \\ &\text{subject to} \quad 0 \leq \alpha_i \leq C, \; i = 1, \ldots, \ell. \end{aligned}$$

The ith component of the gradient of $W(\ )$ is

$$\frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i} = 1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

so one can maximise $W(\ )$ simply by iterating the update rule

$$\alpha_i \leftarrow \alpha_i + \eta \frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i}$$

with a suitable learning rate , and maintaining simultaneously all the $_i$ in the positive orthant. This can be enforced by resetting $_i$ 0 if it becomes negative, that is performing the update

$$\alpha_i \leftarrow \max\left(0, \alpha_i + \eta \frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i}\right).$$

Similarly, the upper bound on the $_i$ due to the soft margin technique can be enforced by setting $_i$ C each time a multiplier becomes larger than C, that is performing the update

$$\alpha_i \leftarrow \min\left(C, \max\left(0, \alpha_i + \eta \frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i}\right)\right),$$

which is also known as the ‘projection method’.

The resulting simple algorithm for training SVMs in the non-bias case is shown in Table 7.1. Note that each training example has been given its own learning rate $_i$.

Table 7.1: Simple on-line algorithm for the 1-norm soft margin

Given training set S and learning rates ($\mathbb{R}$)

$\boldsymbol{\alpha} \leftarrow \mathbf{0}$
repeat
  for $i = 1$ to ℓ
    $\alpha_i \leftarrow \alpha_i + \eta_i \left(1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)\right)$
    if $\alpha_i < 0$ then $\alpha_i \leftarrow 0$
    else
    if $\alpha_i > C$ then $\alpha_i \leftarrow C$
  end for
until stopping criterion satisfied
return $\boldsymbol{\alpha}$

The algorithm fails to implement strict gradient ascent in two ways. Firstly, using a different learning rate for each training example biases the gradient direction. Secondly, if we wish to implement the gradient strictly, we should create a new vector ^{new} inside the *for* loop and then set = ^{new} after completion of the loop. In practice it is convenient and efficient to use the new values of $_i$ as soon as they have been obtained. This approach is known as stochastic gradient ascent and is related to successive overrelaxation techniques for solving matrix equations. As a stopping criterion one can use any of the methods given above, namely monitoring the Karush-Kuhn-Tucker conditions, or the feasibility gap, or simply the rate of increase of the objective function $W(\ )$. Indeed, the stationarity conditions for the algorithm correspond to the KKT conditions of the problem. It is possible to show that for suitable choices of $_i$ the algorithm converges. More precisely, since we are optimising a quadratic function of $_i$ the update that causes the derivative to go to zero is

$$\hat{\alpha}_i \leftarrow \alpha_i + \frac{1}{K(\mathbf{x}_i, \mathbf{x}_i)} \left(1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)\right)$$

since

$$\begin{aligned}\frac{\partial W(\hat{\boldsymbol{\alpha}})}{\partial \alpha_i} &= 1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ &\quad - y_i y_i K(\mathbf{x}_i, \mathbf{x}_i) \frac{1}{K(\mathbf{x}_i, \mathbf{x}_i)} \left(1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)\right) \\ &= 0.\end{aligned}$$

Hence, provided $0 \leq \hat{\alpha}_i \leq C$ the maximal gain is made by choosing

$$\eta_i = \frac{1}{K(\mathbf{x}_i, \mathbf{x}_i)},$$

while a sufficient condition for convergence is $0 < {}_iK(\mathbf{x}_i, \mathbf{x}_i) < 2$. When the update is constrained by the boundary of the feasible region, the step length is shorter and the gain correspondingly smaller, but still positive. Despite this apparent unreliability it can be shown that using the above value for ${}_i$ there exist constants μ, , (0, 1), and τ (0, 1), such that

$$\left\|\boldsymbol{\alpha}^t - \boldsymbol{\alpha}^*\right\| \leq \mu \delta^t,$$

and

$$W\left(\boldsymbol{\alpha}^*\right) - W\left(\boldsymbol{\alpha}^{t+1}\right) \leq \tau \left(W\left(\boldsymbol{\alpha}^*\right) - W\left(\boldsymbol{\alpha}^t\right)\right).$$

Such rates of convergence are reassuring, though in practice the performance of the algorithm can vary very significantly. One important way in which the convergence rate can be improved is to exploit the freedom to vary the order in which the parameters are updated. Table 7.1 shows the parameters being updated sequentially but clearly the order can be varied from one cycle to the next or indeed the update can be made iteratively on any point provided points are not overlooked. Clearly, if we can identify points that will give significant increases to the dual objective function, it will be preferable to choose them first. This suggests the heuristic of choosing from among those points that violate the Karush–Kuhn–Tucker conditions. The outer loop of the algorithm goes through the training set looking for points that violate KKT conditions and selects any it finds for update. In order to increase the chances of finding KKT violations, the outer loop first considers the points for which the corresponding parameter ${}_i$ satisfies $0 < {}_i < C$ implying that its value is not on the boundary of the feasible region, and only when all such points satisfy the KKT conditions to the specified tolerance level is a complete loop through all the training set again undertaken. A simpler variant of this approach is described in Remark 7.8.

Remark 7.5

The algorithm which optimises one dual variable at a time is known in the optimisation literature as Hildreth's method, while in machine learning it is often referred to as kernel-Adatron, because ignoring the use of kernels it is equivalent to the Adatron algorithm for training single neurons.

Remark 7.6

A recently developed variant of the on-line gradient algorithm uses an additional parameter ω (0, 2) to implement successive over-relaxation, giving the update

$$\alpha_i \leftarrow \alpha_i + \frac{\omega}{K(\mathbf{x}_i, \mathbf{x}_i)} \left(1 - y_i \sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j)\right)$$

before clipping, though in the experiments with this approach ω was chosen equal to 1, equivalent to choosing $_i = (K(\mathbf{x}_i, \mathbf{x}_i))^{-1}$, in the algorithm of Table 7.1.

Remark 7.7

One can reduce the training time by caching the values of the kernel matrix $K(\mathbf{x}_i, \mathbf{x}_j)$ during initialisation, since these values are used repeatedly and the kernel can be a costly function to evaluate. This approach to a certain extent sacrifices the inherently on-line character of the algorithm, but is practical if the training set size is moderate. It is adopted in most of the standard techniques described in the following sections, sometimes in conjunction with the subset-selection methods discussed in Section 7.4. For larger datasets one can re-evaluate the kernels at every iteration, reducing the space complexity but increasing the running time. Stochastic gradient ascent is typically used in practice for moderate sample sizes (see Chapter 8 for an example) though, combined with sample selection heuristics, it can also prove effective for massive datasets where other algorithms would fail anyway. We describe one heuristic that proved successful for datasets of one million points in the next remark.

Remark 7.8

Heuristics for choosing the order in which the data points are processed can have a very significant impact on the rate of convergence. A particularly simple ranking is to use the size of the parameters $_i$. This is combined with the two-stage strategy described above of optimising on the current support vectors and, only when this optimum is reached, again considering points with $_i = 0$. This suggests only holding those parameters with non-zero $_i$ in memory. These are the current estimates of the support vectors. They are repeatedly processed in order of ascending size of $_i$ until no further progress is observed in the objective. At this point an iteration through the rest of the dataset is undertaken, which produces a new set of support vectors to begin the cycle over again. We give references in Section 7.8 to papers reporting the successful application of this method to solving the Support Vector optimisation of one million points.

Remark 7.9

Notice that the perceptron algorithm described in Chapter 2 can be regarded as a gradient descent algorithm. The cost function is $\sum_{i=1}$ $_i$ where $_i$ is defined as max $(0, -y_i((\mathbf{w} \cdot \mathbf{x}_i) + b))$. This function is sometimes referred to as the hinge loss and is equal to the linear -insensitive loss with = 0. An inclusive theory of gradient descent algorithms has been developed in recent years, see Section 7.8 for more information.

Remark 7.10

The algorithm described above solves the 1-norm soft margin optimisation problem. Clearly the maximal margin algorithm is recovered if we ignore the $_i \leq C$ constraint. If we wish to solve the 2-norm soft margin optimisation, we simply add a diagonal shift $\mathbf{K} \leftarrow \mathbf{K} + \frac{1}{C}\mathbf{I}$ to the kernel matrix.

Of course the simple procedures outlined in this section suffer from many of the problems associated with naive gradient ascent: they can be extremely slow on some datasets; they can oscillate before converging; and so on. However, their conceptual and computational simplicity makes them ideal candidates for a first implementation of Support Vector Machines, and also for small size applications. More advanced techniques are discussed in the following sections.

6 Chapter 7: Implementation Techniques

7.3 General Techniques and Packages

A number of optimisation techniques have been devised over the years, and many of them can be directly applied to quadratic programmes. The Newton method, conjugate gradient, primal dual interior-point methods, not only can be straightforwardly applied to the case of Support Vector Machines, but given the specific structure of the objective function also can be considerably simplified. Conceptually they are not very different from the simple gradient ascent strategy described above, as they all iteratively climb the objective function to its maximum, but they are all likely to be more efficient, essentially because the direction and the length of each step are chosen in a more sophisticated way. Some care should however be paid to the computational issues arising from the size of the problem. Many of them require that the kernel matrix is stored in memory, implying that the space complexity is quadratic in the sample size. For large size problems, these approaches can be inefficient, and should therefore be used in conjunction with the decomposition techniques described in Section 7.4. As it is impossible to describe the plethora of methods in this chapter, pointers to surveys will be given in Section 7.8.

One of the main advantages of such techniques is that they are well understood, and widely available in a number of commercial and freeware packages, some also accessible through the Internet. It is these packages that were used for Support Vector Machines before specially tailored algorithms were developed. One of the most common choices is the package MINOS, from the Stanford Optimization Laboratory, which uses a hybrid strategy; another standard choice is LOQO, which uses a primal dual interior-point method. In contrast, the quadratic programme subroutine provided in the MATLAB optimisation toolbox is very general but the routine *quadprog* is significantly better than *qp*. Pointers to these and other packages are provided in Section 7.8.

Finally, a very convenient solution is to use one of the existing Support Vector packages, like SVM^{light} by Joachims, the package of Royal Holloway, University of London, and the others freely available through the Internet. Similarly, packages for Gaussian processes are freely available on the Internet. Pointers to literature and on-line software are provided in Section 7.8.

7.4 Chunking and Decomposition

Techniques like those described in the previous section do not have the on-line flavour of stochastic gradient ascent as they require that the data are held in memory in the form of the kernel matrix. The complexity of the training problem grows with the size of this matrix, limiting the approaches to datasets of a few thousand points.

For larger problems we wish to take advantage of an approach that forms the basis of the so-called ‘active set’ or ‘working set’ methods in optimisation: if one knew in advance which constraints were active, it would be possible to discard all of the inactive constraints and simplify the problem. This leads to several strategies, all based on somehow guessing the active set, and restricting training to this guess. Iterative heuristics that gradually build the active set are the most common. Although these techniques are very often heuristics, the fact that at each step they reduce the feasibility gap or increase the dual objective function can be used to guarantee the eventual convergence of the algorithm.

The simplest heuristic is known as *chunking*. It starts with an arbitrary subset or ‘chunk’ of the data, and trains an SVM using a generic optimiser on that portion of the data, and trains an SVM using a generic optimiser on that portion of the data. The algorithm then retains the support vectors from the chunk while discarding the other points and then it uses the hypothesis found to test the points in the remaining part of the data. The M points that most violate the KKT conditions (where M is a parameter of the system) are added to the support vectors of the previous problem, to form a new chunk. This procedure is iterated, initialising for each new sub-problem with the values output from the previous stage, finally halting when some stopping criterion is satisfied. The chunk of data being optimised at a particular stage is sometimes referred to as the working set. Typically the working set grows, though it can also decrease, until in the last iteration the machine is trained on the set of support vectors representing the active constraints. Pseudocode for this algorithm is given in Table 7.2.

Table 7.2: Pseudocode for the general working set method

Given training set S

$\boldsymbol{\alpha} = \mathbf{0}$

select an arbitary working set $\hat{S} \subset S$

repeat

solve optimisation problem on $\hat{S}$

select new working set from data not satisfying Karush–Kuhn–Tucker conditions

until stopping criterion satisfied

return

This heuristic assumes that the kernel matrix for the set of support vectors fits in memory and can be fed to the optimisation package being used. In general, it can happen that the problem is not sparse, or simply that the size is so large that the set of support vectors is still too large to be dealt with by the optimisation routine. One can still deal with such problems by using the more advanced *decomposition* algorithm, which was inspired by the use of chunking in working set methods. The decomposition algorithm only updates a fixed size subset of multipliers α_i, while the others are kept constant. So every time a new point is added to the working set, another point has to be removed. In this algorithm, the goal is not to identify all of the active constraints in order to run the optimiser on all of them, but is rather to optimise the global problem by only acting on a small subset of data at a time. Also in this system, as in the previous one, the ‘nucleus’ of

the algorithm is provided by some generic quadratic programme optimiser, possibly one of the many available packages.

Although no theoretical proof has been given of the convergence of these methods, in practice they work very well and make it possible to deal with datasets of several tens of thousands of points.

The important point is to select the working set in such a way that the optimisation of the corresponding quadratic programme sub-problem leads to an improvement in the overall objective function. Several heuristics for this can be used. An efficient heuristic for choosing the working set at each step is to use the stopping criteria: for example one could include the points that contribute most to the feasibility gap or equivalently that most violate the Karush–Kuhn–Tucker conditions.

7.5 Sequential Minimal Optimisation (SMO)

The Sequential Minimal Optimisation (SMO) algorithm is derived by taking the idea of the decomposition method to its extreme and optimising a minimal subset of just two points at each iteration. The power of this technique resides in the fact that the optimisation problem for two data points admits an analytical solution, eliminating the need to use an iterative quadratic programme optimiser as part of the algorithm.

The requirement that the condition $\sum_{i=1}$ $_i y_i = 0$ is enforced throughout the iterations implies that the smallest number of multipliers that can be optimised at each step is 2: whenever one multiplier is updated, at least one other multiplier needs to be adjusted in order to keep the condition true.

At each step SMO chooses two elements $_i$ and $_j$ to jointly optimise, finds the optimal values for those two parameters given that all the others are fixed, and updates the vector accordingly. The choice of the two points is determined by a heuristic, while the optimisation of the two multipliers is performed analytically.

Despite needing more iterations to converge, each iteration uses so few operations that the algorithm exhibits an overall speed-up of some orders of magnitude. Besides convergence time, other important features of the algorithm are that it does not need to store the kernel matrix in memory, since no matrix operations are involved, that it does not use other packages, and that it is fairly easy to implement. Notice that since standard SMO does not use a cached kernel matrix, its introduction could be used to obtain a further speed-up, at the expense of increased space complexity.

7.5.1 Analytical Solution for Two Points

Without loss of generality we will assume that the two elements that have been chosen are $_1$ and $_2$. In order to compute the new values for these two parameters, one can observe that in order not to violate the linear constraint $\sum_{i=1}$ $_i y_i = 0$, the new values of the multipliers must lie on a line,

$$\alpha_1 y_1 + \alpha_2 y_2 = \text{ constant } = \alpha_1^{old} y_1 + \alpha_2^{old} y_2,$$

in ($_1$, $_2$) space, and in the box defined by $0 \leq$ $_1$, $_2 \leq C$. The one dimensional problem resulting from the restriction of the objective function to such a line can be solved analytically.

Without loss of generality, the algorithm first computes $_2^{new}$ and successively uses it to obtain $_1^{new}$. The box constraint $0 \leq$ $_1$, $_2 \leq C$, together with the linear equality constraint, provides a more restrictive constraint on the feasible values for $_2^{new}$:

$$U \leq \alpha_2^{new} \leq V$$

where

$$\begin{aligned} U &= \max\left(0, \alpha_2^{old} - \alpha_1^{old}\right), \\ V &= \min\left(C, C - \alpha_1^{old} + \alpha_2^{old}\right), \end{aligned} \tag{7.5}$$

if $y_1 \neq y_2$, and

$$\begin{aligned} U &= \max\left(0, \alpha_1^{old} + \alpha_2^{old} - C\right), \\ V &= \min\left(C, \alpha_1^{old} + \alpha_2^{old}\right), \end{aligned} \tag{7.6}$$

if $y_1 = y_2$.

Remark 7.11

In the following theorem we will use the above definitions of U and V. We also introduce some more notation that will simplify the statement and proof of the theorem. We use $f(\mathbf{x})$ to denote the current hypothesis

determined by the values of and b at a particular stage of learning. Let

$$E_i = f(\mathbf{x}_i) - y_i = \left(\sum_{j=1}^{\ell} \alpha_j y_j K(\mathbf{x}_j, \mathbf{x}_i) + b\right) - y_i, \; i = 1, 2, \qquad (7.7)$$

be the difference between function output and target classification on the training points $\mathbf{x}_1$ or $\mathbf{x}_2$. Note that this may be large even if a point is correctly classified. For example if $y_1 = 1$, and the function output is $f(\mathbf{x}_1) = 5$, the classification is correct, but $E_1 = 4$. A further quantity that we will require is the second derivative of the objective function along the diagonal line, which can be expressed as −κ, where

$$\kappa = K(\mathbf{x}_1, \mathbf{x}_1) + K(\mathbf{x}_2, \mathbf{x}_2) - 2K(\mathbf{x}_1, \mathbf{x}_2) = \|\phi(\mathbf{x}_1) - \phi(\mathbf{x}_2)\|^2, \qquad (7.8)$$

where (·) is the mapping into the feature space.

It is now possible to prove the following theorem.

Theorem 7.12

The maximum of the objective function for the optimisation problem (7.1), when only $_1$ and $_2$ are allowed to change, is achieved by first computing the quantity

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{\kappa}$$

and clipping it to enforce the constraint $U \leq {}_2{}^{new} \leq V$:

$$\alpha_2^{new} = \begin{cases} V, & if\ \alpha_2^{new,unc} > V, \\ \alpha_2^{new,unc}, & if\ U \leq \alpha_2^{new,unc} \leq V, \\ U, & if\ \alpha_2^{new,unc} < U, \end{cases}$$

where E_i is given in equation (7.7), κ is given by equation (7.8), and U and V are given by equation (7.5) or (7.6). The value of $_1{}^{new}$ is obtained from $_2{}^{new}$ as follows:

$$\alpha_1^{new} = \alpha_1^{old} + y_1 y_2(\alpha_2^{old} - \alpha_2^{new}).$$

Proof Define

$$v_i = \sum_{j=3}^{\ell} y_j \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) = f(\mathbf{x}_i) - \sum_{j=1}^{2} y_j \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - b, \;\; i = 1, 2,$$

and consider the objective as a function of $_1$ and $_2$:

$$\begin{aligned} W(\alpha_1, \alpha_2) \;=\; & \alpha_1 + \alpha_2 - \frac{1}{2}K_{11}\alpha_1^2 - \frac{1}{2}K_{22}\alpha_2^2 \\ & - y_1 y_2 K_{12}\alpha_1\alpha_2 - y_1\alpha_1 v_1 - y_2\alpha_2 v_2 + \text{constant}, \end{aligned}$$

where $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, $i, j = 1,2$. Note also that the constraint $\sum_{i=1} {}_i{}^{old} y_i = \sum_{i=1} {}_i y_i = 0$ implies the condition

$$\alpha_1 + s\alpha_2 = \text{constant} = \alpha_1^{old} + s\alpha_2^{old} = \gamma,$$

where $s = y_1 y_2$. This equation demonstrates how $_1{}^{new}$ is computed from $_2{}^{new}$. The objective function along the constraint becomes

$$\begin{aligned} W(\alpha_2) \;=\; & \gamma - s\alpha_2 + \alpha_2 - \frac{1}{2}K_{11}(\gamma - s\alpha_2)^2 - \frac{1}{2}K_{22}\alpha_2^2 \\ & - sK_{12}(\gamma - s\alpha_2)\alpha_2 - y_1(\gamma - s\alpha_2)v_1 - y_2\alpha_2 v_2 + \text{constant} \end{aligned}$$

and the stationary point satisfies

$$\begin{aligned}\frac{\partial W(\alpha_2)}{\partial \alpha_2} &= 1 - s + sK_{11}(\gamma - s\alpha_2) - K_{22}\alpha_2 \\ &\quad + K_{12}\alpha_2 - sK_{12}(\gamma - s\alpha_2) + y_2 v_1 - y_2 v_2 \\ &= 0.\end{aligned}$$

This yields

$$\begin{aligned}\alpha_2^{new,unc}(K_{11} + K_{22} - 2K_{12}) &= 1 - s + \gamma s(K_{11} - K_{12}) + y_2(v_1 - v_2) \\ &= y_2(y_2 - y_1 + \gamma y_1(K_{11} - K_{12}) + v_1 - v_2).\end{aligned}$$

Hence,

$$\begin{aligned}\alpha_2^{new,unc}\kappa y_2 &= y_2 - y_1 + f(\mathbf{x}_1) - \sum_{j=1}^{2} y_j \alpha_j K_{1j} + \gamma y_1 K_{11} \\ &\quad - f(\mathbf{x}_2) + \sum_{j=1}^{2} y_j \alpha_j K_{2j} - \gamma y_1 K_{12} \\ &= y_2 - y_1 + f(\mathbf{x}_1) - f(\mathbf{x}_2) \\ &\quad + y_2\alpha_2 K_{11} - y_2\alpha_2 K_{12} + y_2\alpha_2 K_{22} - y_2\alpha_2 K_{12} \\ &= y_2\alpha_2\kappa + (f(\mathbf{x}_1) - y_1) - (f(\mathbf{x}_2) - y_2),\end{aligned}$$

giving

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{y_2(E_1 - E_2))}{\kappa}.$$

Finally, we must clip $_2^{new,unc}$ if necessary to ensure it remains in the interval [*U, V*].

7.5.2 Selection Heuristics

In order to speed up the convergence, it is possible to choose the set of two points to optimise, based on their contribution to the general progress towards the solution. If the amount of computation required to implement the selection strategy is smaller than that saved by the reduction in the number of iterations, one obtains a gain in the rate of convergence.

The chosen stopping criterion can give a good indication of which points are more likely to make a larger contribution towards convergence. For example, if one monitors the feasibility gap, a natural choice is to optimise the points that most violate the KKT conditions, as they contribute most to that gap (see stopping criterion 3 in Section 7.1). Computing the KKT conditions of each point at each iteration is, however, computationally expensive, and cheaper heuristics can deliver a better overall performance.

SMO uses two criteria for selecting the two active points to ensure that the objective function enjoys a large increase from their optimisation. There are two separate heuristics for choosing the first and the second point.

- **First Choice Heuristic** The first point $\mathbf{x}_1$ is chosen from among those points that violate the Karush–Kuhn–Tucker conditions. The outer loop of the algorithm goes through the training set looking for points that violate KKT conditions and selects any it finds for update. When one such point is found, the second heuristic is used to select the second point, and the values of the respective multipliers are updated. Then the outer loop is resumed looking for new KKT violations. In order to increase the chances of finding KKT violations, the outer loop goes through the points for which the corresponding parameter $_i$ satisfies $0 < {}_i < C$ implying that its value is not on the boundary of the feasible region, and only when all such points satisfy the KKT conditions to the specified tolerance level is a complete loop through all the training set again undertaken.

- **Second Choice Heuristic** The second point $\mathbf{x}_2$ must be chosen in such a way that updating on the pair $_1$, $_2$ causes a large change, which should result in a large increase of the dual objective. In order to find a good point without performing too much computation, a quick heuristic is to choose $\mathbf{x}_2$ to maximise the quantity ∣$E_1 - E_2$∣, where E_i is defined in Theorem 7.12. If E_1 is positive, SMO chooses an example $\mathbf{x}_2$ with minimum error E_2, while if E_1 is negative, SMO maximises the error E_2. A cached list of errors for every non-bound point in the training set is kept to further reduce the computation. If this choice fails to deliver a significant increase in dual objective, SMO tries each non-bound point in turn. If there is still no significant progress SMO looks through the entire training set for a suitable point. The loops through the non-bound points, and the whole training set, start from random locations in the respective lists, so that no bias is introduced towards the examples occurring at the beginning of either of them.

We have included pseudocode for the SMO classification algorithm due to John Platt in Appendix A. Note also that in Section 7.8, pointers are provided to on-line software implementing SMO, as well as to references with a complete and detailed description of SMO.

Remark 7.13

Note that the SMO algorithm makes apparent use of the parameter C, and so at first appears only applicable to the 1-norm soft margin optimisation problem. We can, however, run SMO treating C as infinite, and hence reducing the constraints on the interval $[U, V]$, which subsequently only provides a lower bound on $_2^{new}$ of

$$U = \max\left(0, \alpha_2^{old} - \alpha_1^{old}\right)$$

when $y_1 \neq y_2$, and is given by

$$\begin{aligned} U &= 0, \\ V &= \alpha_1^{old} + \alpha_2^{old} \end{aligned}$$

if $y_1 = y_2$.

Remark 7.14

The SMO algorithm makes no provision for choosing the bias, and yet uses it in calculating the values E_i, $i = 1,2$. This apparent indeterminacy does not change the algorithm since whatever value of b is used both E_iS are equally affected and so the resulting update, being determined by their difference, is identical. Hence, b can be taken to be zero throughout the computation and set after convergence using the Karush–Kuhn–Tucker conditions as described at the beginning of this chapter for the particular optimisation being performed. Note, however, that we may need to compute b in order to evaluate the stopping criterion.

Remark 7.15

The stopping criterion 3 in Section 7.1 can be used to assess convergence. Note that this will require setting the bias. As indicated as part of that calculation the b should be chosen by reference to some $_i$ satisfying $0 < _i < C$. The original SMO algorithm calculated the bias based on an estimate derived from the updated points. If neither satisfies the required inequalities, this could lead to an over-estimate of the feasibility gap.

Remark 7.16

Notice that SMO is not directly applicable in the fixed bias case, since the choice of $_2^{new}$ was made using the constraint resulting from the variable bias. For fixed bias the SMO algorithm reduces to the algorithm described in Table 7.1 with

$$\eta_i = (K(\mathbf{x}_i, \mathbf{x}_i))^{-1}.$$

For regression the SMO algorithm can again update α_1 and α_2 from the optimisation problem given in problem (7.2). The equations encode four separate problems depending on the signs of the two parameters. Here, the constraint on the α vector does not involve the classifications and so the interval for α_2 is given by

$$\begin{aligned} U &= \max\left(C_U^2, \alpha_1^{old} + \alpha_2^{old} - C_V^1\right), \\ V &= \min\left(C_V^2, \alpha_1^{old} + \alpha_2^{old} - C_U^1\right), \end{aligned} \tag{7.9}$$

where the four problems each have different settings of the parameters $C^i{}_U$ and $C_V{}^i$ that are specified in the following table:

	$\alpha_i \geq 0$	$\alpha_i \leq 0$
C_U^i	0	$-C$
C_V^i	C	0

(7.10)

Remark 7.17

In the following theorem we will use the above definitions of U and V. We also introduce some more notation that will simplify the statement and proof of the theorem. We use $f(\mathbf{x})$ to denote the current hypothesis determined by the values of α and b at a particular stage of learning. Let

$$E_i = f(\mathbf{x}_i) - y_i = \left(\sum_{j=1}^{\ell} \alpha_j K(\mathbf{x}_j, \mathbf{x}_i) + b\right) - y_i, \; i = 1, 2, \tag{7.11}$$

be the difference between function output and target value on the training points $\mathbf{x}_1$ or $\mathbf{x}_2$.

It is now possible to prove the following theorem showing how to apply SMO for the regression case. Note that this theorem is effectively giving four update rules, one for each quadrant of the space of current values of α_1 and α_2. It is important that the optimisation is performed and hence the new values are sought within a chosen quadrant containing the current values. If the current values fall in more than one quadrant then the corresponding optimisations can be performed for each quadrant and the one giving greater increase in the objective function chosen.

Theorem 7.18

The maximum of the objective function for the optimisation problem (7.1), when only α_1 and α_2 are allowed to change in a specified quadrant containing them both, is achieved by first computing the quantity

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{(E_1 - E_2) - \varepsilon(\operatorname{sgn}(\alpha_2) - \operatorname{sgn}(\alpha_1))}{\kappa}$$

and clipping it to enforce the constraint $U \leq \alpha_2{}^{new} \leq V$:

$$\alpha_2^{new} = \begin{cases} V, & \text{if } \alpha_2^{new,unc} > V, \\ \alpha_2^{new,unc}, & \text{if } U \leq \alpha_2^{new,unc} \leq V, \\ U, & \text{if } \alpha_2^{new,unc} < U, \end{cases}$$

where the values of the sgn function are determined by the chosen quadrant, E_i is given in equation (7.11), κ is given by equation (7.8), and U and V are given by equation (7.9) or (7.10). The value of $\alpha_1{}^{new}$ is obtained from $\alpha_2{}^{new}$ as follows:

$$\alpha_1^{new} = \alpha_1^{old} + \alpha_2^{old} - \alpha_2^{new}.$$

Proof We define

$$v_i = \sum_{j=3}^{\ell} \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) = f(\mathbf{x}_i) - \sum_{j=1}^{2} \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) - b, \; i = 1, 2,$$

and consider the objective as a function of α_1 and α_2:

$$\begin{aligned} W(\alpha_1, \alpha_2) &= y_1\alpha_1 + y_2\alpha_2 - \varepsilon\left(|\alpha_1| + |\alpha_2|\right) - \frac{1}{2}K_{11}\alpha_1^2 - \frac{1}{2}K_{22}\alpha_2^2 \\ &\quad -K_{12}\alpha_1\alpha_2 - \alpha_1 v_1 - \alpha_2 v_2 + \text{constant}, \end{aligned}$$

where $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, $ij = 1,2$. Note also that the constraint $\sum_{i=1} \alpha_i^{old} = \sum_{i=1} \alpha_i = 0$ implies the condition

$$\alpha_1 + \alpha_2 = \text{constant} = \alpha_1^{old} + \alpha_2^{old} = \gamma.$$

This equation demonstrates how α_1^{new} is computed from α_2^{new}. The objective function along the constraint becomes

$$\begin{aligned} W(\alpha_2) &= y_1\gamma - y_1\alpha_2 + y_2\alpha_2 - \varepsilon\left(|\alpha_1| + |\alpha_2|\right) - \frac{1}{2}K_{11}(\gamma - \alpha_2)^2 - \frac{1}{2}K_{22}\alpha_2^2 \\ &\quad -K_{12}(\gamma - \alpha_2)\alpha_2 - (\gamma - \alpha_2)v_1 - \alpha_2 v_2 + \text{constant} \end{aligned}$$

and the stationary point satisfies

$$\begin{aligned} \frac{\partial W(\alpha_2)}{\partial \alpha_2} &= y_2 - y_1 - \varepsilon(\text{sgn}(\alpha_2) - \text{sgn}(\alpha_1)) + K_{11}(\gamma - \alpha_2) - K_{22}\alpha_2 \\ &\quad +K_{12}\alpha_2 - K_{12}(\gamma - \alpha_2) + v_1 - v_2 \\ &= 0. \end{aligned}$$

This yields

$$\begin{aligned} \alpha_2^{new,unc}(K_{11} + K_{22} - 2K_{12}) &= y_2 - y_1 - \varepsilon(\text{sgn}(\alpha_2) - \text{sgn}(\alpha_1)) \\ &\quad + \gamma(K_{11} - K_{12}) + v_1 - v_2 \\ &= y_2 - y_1 + \gamma(K_{11} - K_{12}) + v_1 - v_2. \end{aligned}$$

Hence,

$$\begin{aligned} \alpha_2^{new,unc}\kappa &= y_2 - y_1 + f(\mathbf{x}_1) - \sum_{j=1}^{2} \alpha_j K_{1j} + \gamma K_{11} \\ &\quad -f(\mathbf{x}_2) + \sum_{j=1}^{2} \alpha_j K_{2j} - \gamma K_{12} - \varepsilon(\text{sgn}(\alpha_2) - \text{sgn}(\alpha_1)) \\ &= y_2 - y_1 + f(\mathbf{x}_1) - f(\mathbf{x}_2) \\ &\quad +\alpha_2 K_{11} - \alpha_2 K_{12} + \alpha_2 K_{22} - \alpha_2 K_{12} - \varepsilon(\text{sgn}(\alpha_2) - \text{sgn}(\alpha_1)) \\ &= \alpha_2\kappa + (f(\mathbf{x}_1) - y_1) - (f(\mathbf{x}_2) - y_2) - \varepsilon(\text{sgn}(\alpha_2) - \text{sgn}(\alpha_1)), \end{aligned}$$

giving

$$\alpha_2^{new,unc} = \alpha_2^{old} + \frac{E_1 - E_2 - \varepsilon(\text{sgn}(\alpha_2) - \text{sgn}(\alpha_1))}{\kappa}.$$

Finally, we must clip $\alpha_2^{new,unc}$ if necessary to ensure it remains in the interval $[U, V]$.

7.6 Techniques for Gaussian Processes

The solution of the Bayesian learning problem for the training set

$$S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)),$$

using Gaussian processes was described in Subsection 6.2.3 where it was shown to be equivalent to ridge regression using the covariance matrix as a kernel. Solving this problem involves computing (see equation (6.10)) the parameter vector satisfying

$$f(\mathbf{x}) = \mathbf{y}'(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k} = \boldsymbol{\alpha}'\mathbf{k} = \sum_{i=1}^{\ell} \alpha_i K(\mathbf{x}_i, \mathbf{x}),$$

where

$$\begin{aligned} \boldsymbol{\alpha} &= (\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{y}, \text{ or} \\ \mathbf{y} &= (\mathbf{K} + \sigma^2\mathbf{I})\boldsymbol{\alpha}, \end{aligned}$$

where $\mathbf{k}$ is a vector whose ith entry is $K(\mathbf{x}_i, \mathbf{x})$ and $\mathbf{K}$ the kernel matrix with entries $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$. Hence, the solution of the problem is obtained by solving a system of linear equations of size × , while the evaluation of the solution on a novel input requires the computation of an inner product involving the values of the kernel between the new point and each of the training examples. Note that in the case of a Gaussian process there is no reason to expect the parameter vector to be sparse.

Several methods exist for solving systems of linear equations, for example LU decomposition, Gauss Jordan elimination and in our case of a symmetric matrix Cholesky decomposition. Most off-the-shelf numerical packages will offer a choice of methods. Unfortunately the complexity of solving a system of linear equations scales with ^{3}, making the approach impractical for very large datasets.

The problem can be tackled by a naive gradient descent method. Indeed Exercise 1 of Chapter 2 asked the reader to rewrite the Widrow–Hoff algorithm in dual form. If this is implemented without a bias and using the augmented kernel $\mathbf{K} + {}^2\mathbf{I}$, it will converge to the solution of the Gaussian process. A more sophisticated way of tackling the computational complexity problems is provided by conjugate gradient techniques. Table 7.3 shows the sequence of calculations that perform the conjugate gradient calculation.

Table 7.3: Pseudocode for the conjugate gradient method

$\boldsymbol{\alpha}^1 \leftarrow \mathbf{0}$, $\mathbf{h}^1 \leftarrow \mathbf{g}^1 \leftarrow \mathbf{y}$
for $k = 1, \ldots, n$
$\quad \lambda \leftarrow \dfrac{(\mathbf{g}^k)'\mathbf{g}^k}{(\mathbf{g}^k)'\mathbf{C}\mathbf{h}^k}$
$\quad \boldsymbol{\alpha}^{k+1} \leftarrow \boldsymbol{\alpha}^k + \lambda\mathbf{h}^k$
$\quad \mathbf{g}^{k+1} \leftarrow \mathbf{g}^k - \lambda\mathbf{C}\mathbf{h}^k$
$\quad \gamma \leftarrow \dfrac{(\mathbf{g}^{k+1})'\mathbf{g}^{k+1}}{(\mathbf{g}^k)'\mathbf{g}^k}$
$\quad \mathbf{h}^{k+1} \leftarrow \mathbf{g}^{k+1} + \gamma\mathbf{h}^k$
end for
return $\boldsymbol{\alpha}$

The process is guaranteed to converge to the solution in $n =$ iterations, but an early stopping will deliver an approximation of $_{\max}$ with a complexity that scales only as $n \times {}^2$. The quality of the approximation can be

estimated using so-called Skilling methods, which hence give a good stopping criterion for the iterative procedure (for more details, follow the links in Section 7.8).

7.7 Exercises

1. Implement the gradient descent algorithm for the maximum margin optimisation problem. Test it on an artificially created dataset. Print out the Karush–Kuhn–Tucker conditions. Plot the margin as a function of the number of iterations. Introduce some sample selection heuristics and redo the plot.
2. Try the same algorithm on non-separable data. Add the soft margin features to the algorithm. Experiment with the tunable parameters. Run the above algorithms on a small real-world dataset, taken for example from the UCI repository [96].
3. Implement SMO for classification and run it on a range of datasets.

7.8 Further Reading and Advanced Topics

The problem of convex optimisation has been extensively studied since the 1950s, and a number of techniques exist to solve it. This chapter did not attempt to be exhaustive, but simply to provide the reader with some techniques that are both easy to implement and exemplify some standard approaches. Discussions of optimisation algorithms can be found in the books [41], [80], [11], and [86].

For the particular case of Support Vector Machines, see the excellent surveys by Smola and Schölkopf [145] and Burges [23]. Implementation problems and techniques are discussed by Kauffman [70], Joachims [68], Platt [114] [112], Osuna and Girosi [111], and Keerthy et al. [73] [74].

The gradient ascent techniques were introduced into the SVM literature by papers on the kernel-Adatron procedure, [44] [24]; similar ideas have been used independently by Haussler and Jaakkola [65]. They are also related to SMO with fixed b [112]; and to Hildreth's QP method [62]. Mangasarian and his co-workers recently introduced algorithms that can deal with massive datasets ([90], [91], [20]). Albeit with different motivations, the algorithm SOR (Successive Over Relaxation) of Mangasarian and Musicant [89] is equivalent to the stochastic gradient ascent algorithm described in Section 7.2, combined with sample selection heuristics motivated by Platt's approach [112]. Mangasarian and Musicant [89] also give the proof of linear convergence of the algorithm using the link with SOR.

In Remark 7.4, we discussed the case when it is possible to use fixed bias; note that also Jaakkola and Haussler [65] use this assumption; and Mangasarian and Musicant [89] discuss the cases in which such an assumption is not too restrictive. Similar approaches have also been used in [170] and [45].

An on-line theory of generalisation for gradient descent algorithms has been developed by Littlestone, Warmuth and others [75]. A discussion of square loss algorithms can be found in [75], while the theory of hinge loss is developed in [48]. This theory provides tight bounds on the maximum number of mistakes an on-line algorithm can make in the worst case, and requires surprisingly few assumptions. The bounds can also be translated for the case where the data have been generated by a distribution. One of its best-known consequences is the motivation of multiplicative updating algorithms, and the characterisation of the cases in which they are expected to perform better than standard gradient descent. A multiplicative algorithm for Support Vector Machines has been studied by Cristianini et al. [29].

The elegant SMO algorithm was devised by Platt [112], and applied to text categorisation problems. Pseudocode (kindly supplied by John Platt) for SMO is available in Appendix A of this book. An extension of SMO, differing in the way it calculates the bias, has been proposed in [74] and shown to be faster. Alex Smola has generalised SMO for the case of regression [148], [145], and the code can be found on the GMD website [53].

Keerthy et al. [73] proposed a very elegant algorithm for SVMs that does not maximise the margin by minimising the norm of the weight vector. Rather, they note that the distance vector between the nearest points of the convex hulls of the positive and negative data uniquely determines the maximal margin hyperplane. This more geometrical view of the problem is discussed for example in [14] and in [129]. Exercise 2 of Chapter 5 asks the reader to devise an optimisation problem for this criterion, hence motivating this alternative solution strategy for the maximum margin problem. Based on the same approach, Kowalczyk [76] has proved a convergence rate for a new iterative algorithm that can be applied to the hard and soft margin problems, as well as extensive experimental comparisons with other iterative algorithms Guyon and Stork [56] give another variant of an iterative algorithm for soft margin optimisation.

Chunking techniques in Support Vector Machines were already used by Vapnik and Chervonenkis, and were improved, generalised and discussed in a number of papers, among others Osuna and Girosi [111], [110], [109], Joachims [68], Platt [112], Smola and Schölkopf [145], and Kauffman [70]. The work of Osuna and Girosi inspired the subsequent work on data selection, which ultimately led to systems like SMO.

Techniques exist for tuning the parameters automatically. For example [31] adjusts the kernel parameters while the ν-SVMs allow the user to set an upper bound on the number of support vectors for classification [135], and regression [130]. The stopping criterion based on the feasibility gap is discussed in [148], who proposes a criterion similar to that of equation (7.3). Other criteria are discussed in [73] and [74].

The implementation of Gaussian processes is discussed in [50]. An experimental study of Widrow–Hoff with kernels can be found in [46].

The book [100] discusses general techniques for convex optimisation and gives pointers to commercial optimisation packages. Early implementations of SVMs have been based on optimisation packages such as MINOS [101], LOQO [156], MATLAB optimisation package [92], and others mentioned above. Chapter 1 of the book [149] contains a useful survey of different implementations.

Packages specifically for SVM implementation are available on-line. The package jointly prepared by the groups at Royal Holloway, ATT and GMD FIRST as available at the Royal Holloway, University of London, website [126]. The package SVM^{light} of Joachims [68] is also available on the web via the website [30]. The package prepared by Alex Smola for SVM classification is available at the GMD-FIRST website [53]. Web links to this and other software are available via the website [30].

The original reference for the conjugate gradient algorithm is [61], while the discussion of Skilling methods for estimating the quality of the approximation obtained is given in [49]. Software is also available on-line for Gaussian processes: Radford Neal's code [103]; Gibbs and MacKay [49].

These reference are also given on the website **http://www.support-vector.net,** which will be kept up to date with new work, pointers to software and papers that are available on-line.

Chapter 8: Applications of Support Vector Machines

Overview

In this chapter we illustrate the use of the algorithms described in this book, by examining some interesting applications. Support Vector Machines have been applied to many real-world problems, so that the material in this chapter is by no means exhaustive. The role of this final chapter is to show how the approach can be used successfully in very diverse fields, and how the necessary design choices can be made in each case.

Applying the Support Vector approach to a particular practical problem involves resolving a number of design questions. We do not dwell on the question of precise specification of the input domain, generation of the training data, and so on. We treat these as given, though in practice they can frequently be refined through an interaction between the SVM system designer and the domain practitioner.

Such a collaboration may also be required to resolve the first design problem, that of choosing an appropriate kernel for the given application. There are standard choices such as a Gaussian or polynomial kernel that are the default options, but if these prove ineffective or if the inputs are discrete structures more elaborate kernels will be needed. By implicitly defining a feature space, the kernel provides the description language used by the machine for viewing the data. Frequently, the designer can work with the more intuitive notion of similarity in the input space and this is where domain experts can give invaluable assistance in helping to formulate appropriate similarity measures.

Typically the choice of kernel will in fact be a family of kernels parametrised by some hyperparameter, as for example in the case of Gaussian kernels where the parameter remains to be determined. Again there may be heuristics for setting these parameters; in the Gaussian case a good choice of is the distance between closest points with different classifications; but frequently the choice must be made in response to the data. In the absence of reliable criteria, applications rely on the use of a validation set or cross-validation to set such parameters.

The next decision facing the SVM system designer is which flavour of SVM to implement. Assuming the problem requires the data to be classified, a decision must be taken whether to use the maximal margin, or one of the soft margin approaches. Here, the critical factors will be the dimensionality of the data and the type and level of noise present in their generation.

Once the choice of kernel and optimisation criterion has been made the key components of the system are in place. Experience with other learning systems might lead one to expect that what follows is a lengthy series of experiments in which various parameters are tweaked until eventually satisfactory performance is achieved. The examples we report in this chapter we believe demonstrate that in many cases the most straightforward SVM implementations outperform other techniques without any need for further adaptation. This experience with SVMs suggests that the range of areas in which they can successfully be applied is far from fully explored.

The applications that we describe are intended to serve as examples of what can be achieved. We therefore give the main thrust of each case study and refer the reader to the relevant papers should they wish to get a detailed description of the approach and results obtained. Our focus is on describing the problem and the specific kernel together with the type of optimisation used in the given application.

We include cases of text (and hypertext) categorisation, image classification, biosequence analysis and biological data mining, hand-written character recognition, etc. We have tended to choose simple examples

that are frequently more illustrative, though we will refer to more complex studies of the same problems in Section 8.5.

8.1 Text Categorisation

The task of text categorisation is the classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content. This problem arises in a number of different areas including email filtering, web searching, office automation, sorting documents by topic, and classification of news agency stories. Since a document can be assigned to more than one category this is not a multi-class classification problem, but can be viewed as a series of binary classification problems, one for each category.

One of the standard representations of text for the purposes of information retrieval (IR) provides an ideal feature mapping for constructing a Mercer kernel. Hence, in this case the use of prior knowledge developed in another field inspires the kernel design; we will see in the following sections that this is a pattern repeated in many case studies. Indeed, the kernels somehow incorporate a similarity measure between instances, and it is reasonable to assume that experts working in the specific application domain have already identified valid similarity measures, particularly in areas such as information retrieval and generative models.

8.1.1 A Kernel from IR Applied to Information Filtering

In the information retrieval literature a common technique for finding text documents is the so-called vector space model, also known as the bag-of-words representation. In this method a document **x** is represented by a vector $\phi(x)$ indexed by a pre-fixed set or dictionary of terms; the value of an entry can be a Boolean variable indicating the presence of the corresponding term, or can indicate the weight of the term within the document in a way we will describe below. Finally, the vectors are normalised to remove information about the length of the text. The distance between two documents is then computed by calculating the inner product between the corresponding vectors. In this model, all information about the order of the words is lost, a situation that resembles the case of image classification discussed below in Section 8.2.

Research in information retrieval has shown that word stems provide good representation units. A word stem is derived from a word by removing case and other inflection information. An example is the words ‘computer’, ‘computation’ and ‘computing’, which all have the same word stem ‘comput’. The dictionary can be chosen by ranking all the word stems occurring in the database according to their mutual information, removing uninformative words (such as ‘and’, etc.) also known as *stop-words*, and selecting the first word stems.

A standard choice for the weighting factor for each word stem is the following:

$$\phi_i(x) = \frac{\mathrm{tf}_i \log(\mathrm{idf}_i)}{\kappa},$$

where tf_i is the number of occurrences of the term i in the document x, idf_i is the ratio between the total number of documents and the number of documents containing the term, and κ is the normalisation constant ensuring that ∣∣ ∣∣$_2$ = 1.

In this way a document is represented by a vector in which each entry records how many times a particular word stem is used in the document weighted according to how informative that word stem is. Typically ϕ can have tens of thousands of entries, often comparable with or even larger than the number of training examples. Furthermore, for a particular document the representation is typically extremely sparse, having only a few non-zero entries. But given that only inner products between the data are needed for the algorithm and the evaluation function, a series of techniques can be exploited to facilitate the computations.

The function $K(x,z) = \phi(x) \cdot \phi(z)$ is clearly a valid kernel, since it is the inner product in an explicitly constructed feature space. Its kernel matrix is therefore always positive semi-definite, and Support Vector Machines can be used with $K(x,z)$ for discrimination. Given the extreme sparseness and the high dimensionality of the vector $\phi(x)$, it is reasonable to assume that the points in the feature space are going to be

easy to separate, and that a standard hard margin SVM will be sufficient. Furthermore, it is possible to devise fast techniques for computing sparse inner products.

The dataset chosen both by Joachims [67] and by Dumais et al. [36] is a collection of labelled Reuters news stories, the Reuters-21578 dataset, compiled by David Lewis from data of Reuters Newswire in 1987, and publicly available. The specific split of the data that was used is called ‘ModApte’ and involves a training set of 9603 documents and a test set of 3299 documents to be classified into 90 categories. After preprocessing by word stemming, and removal of the stop-words, the corpus contains 9947 distinct terms that occur in at least three documents. The stories are on average about 200 words long. Examples of categories in this dataset are corporate acquisitions, earning, money market, corn, wheat, ship, and so on, each with a different number of examples. Note that many documents are assigned to several categories.

More experiments were performed by Joachims on the Oshumed (Oregon Health Sciences University) corpus, compiled in 1991 by William Hersh, containing 50216 documents with abstracts, the first 10000 being used for training, and the second 10000 for testing. The task is to assign each document to one or more of 23 categories representing diseases. After preprocessing (stemming and removal of stop-words), the training corpus contains 15561 distinct terms that occur in at least three documents.

In Joachims' experiments, a simple maximal margin SVM was used, with a heuristic for removing documents whose $_i$ grew too large and that were therefore regarded as outliers. This strategy was able to replace the need for soft margin techniques because of the particular representation used for the data. Polynomial kernels of low degree and Gaussian kernels have been tested with very positive results. Independently of the choice of kernel parameter, the SVM performed better than standard methods such as naive Bayes, Rocchio, the decision tree algorithm C4.5, and the *k*-nearest neighbour method.

Remark 8.1

The training algorithm used in these experiments was SVM^{light}, developed by Joachim and freely available over the Internet (see Section 7.8).

Remark 8.2

It is interesting to ask why SVMs were particularly effective in this application. In traditional systems, when the number of features is higher than the training set size, one expects bad generalisation in the same way as was suggested in Chapter 4 for linear functions in high dimensional spaces. This problem can only be overcome if the learning algorithm is able to take advantage of the benign relationship between the target function and the distribution. The SVM maximisation of the margin optimises this advantage, hence overcoming the difficulties inherent in very high dimensional representations.

Remark 8.3

The features used by this approach are low level ones obtained by computing the word frequency in the documents. They only capture global properties of the document. The problem of using more sophisticated kernels with higher level features remains open. A first step would be to consider the string kernel introduced in Example 3.15 in Chapter 3. This kernel will not take into account the order in which the words or substrings occur and is unable to be selective about which words are included, but it works in a much higher dimensional space, this being possible as the feature vector is no longer computed explicitly.

8.2 Image Recognition

Large collections of digital images are becoming freely available over the Internet, or in specialised databases. Furthermore the generation of images has become extremely cheap, and is exploited in several applications. The automatic categorisation of images is a crucial task in many application areas, including Information Retrieval, filtering Internet data, medical applications, object detection in visual scenes, etc. Much of the research in image categorisation focuses on extraction of high level features from images, for example using edge detection and shape description techniques, in order to capture relevant attributes of the image without increasing the number of features by too much. However, quick information retrieval techniques have also been developed that use only low level features.

Traditional classification approaches perform poorly when working directly on the image because of the high dimensionality of the data, but Support Vector Machines can avoid the pitfalls of very high dimensional representations, as already demonstrated for the case of text in the previous section. A very similar approach to the techniques described for text categorisation can also be used for the task of image classification, and as in that case linear hard margin machines are frequently able to generalise well. In image retrieval tasks, low level features such as histograms of the luminosity or colour levels are often used. They capture global low level properties of the image, conceptually analogous to those used in the text categorisation example. Like word stems they also fail to retain positional information. The distance between two such histograms can be estimated with a χ^2 or other measure of discrepancy between distributions.

In other cases an image is represented directly by its bitmap matrix, which is also the representation used for hand-written digit recognition described in Section 8.3. We begin by considering experiments using this representation.

The use of more sophisticated kernels is largely unexplored and likely to deliver better results. Interaction with practitioners should help in the design of such kernels.

8.2.1 Aspect Independent Classification

A study of the performance of SVMs in these conditions was performed by Pontil and Verri [118], who used SVMs for aspect independent object recognition. They used the simplest way to represent an image as a matrix of bits, or bitmap. Matrices can then be regarded as vector inputs. If the matrices have height h and width w, colour images become vectors of length $3 \times h \times w$, while grey level images have length $h \times w$. Each component of the vector is a pixel in a specific position.

The benchmarking was performed on the COIL (Columbia Object Image Library, available over the Internet) dataset, comprising 7200 images: 72 different views of 100 different three dimensional objects, where each set of 72 views corresponded to 5 degree rotations of the same object. They were transformed into grey level pictures, and their resolution was reduced from 128×128 pixels to 32×32 pixels, by averaging patches of 4×4 pixels. After this preprocessing phase, each image of the dataset was transformed into a vector of 1024 components, each grey level represented by an eight bit number. A simple linear inner product between these vectors was chosen for the kernel, and the basic maximal margin classifier was implemented.

Note that the system does not perform high level feature extraction, but classifies images by viewing them as points in the high dimensional pixel space. The 7200 images were divided into two groups of 3600 images for training and testing, each group containing one of the subset of 10 degree rotations of each of the 100 images. The inherent multi-class nature of the problem was addressed by training a classifier for each couple of classes and combining them in a special way (see the paper for details). For each experiment, however, only a random subset of 32 of the 100 classes was used. The system was trained on 36 views of the three dimensional objects, and tested on the other 36 views of the objects.

Given the high dimensionality of the feature representation, hard margin hyperplanes with no soft margin and no kernels were sufficient to exactly separate the data. This was also sufficient to always achieve an extremely high performance on the test set. In many cases, only the addition of artificial noise to the data could induce the machine to mistake the predictions. In comparison, standard perceptrons trained in the same conditions exhibited a much worse performance.

8.2.2 Colour-Based Classification

The experiments of [118] relied entirely on pixel grey level information, and neglected important sources of information such as colours. This information can be exploited for classification with SVMs, as demonstrated in a series of experiments by Olivier Chapelle and co-workers [25] that only used colour and luminescence information.

Their approach uses features that somewhat resemble the ones described in the text categorisation section. Based on established information retrieval techniques, they use as a similarity measure between images a distance between their respective histograms of colours. Unfortunately, they do not prove that such kernels satisfy Mercer's conditions. However, it is particularly interesting and illustrative to see domain-specific knowledge used to define a kernel, subsequently applied in a Support Vector Machine. Indeed the same pattern will also be found in the biosequences example.

The kernels are specified by choosing a description of the images, technically a map from images to feature vectors, and a similarity measure on the feature vectors. As discussed above, the simplest way to represent an image is as a matrix of bits, or bitmap. This representation, however is not scale and translation invariant. A slightly more sophisticated representation makes use of histograms of colour. Each colour is a point in a three dimensional colour space. Each image is associated to a histogram, encoding the fraction of pixels of a given colour. The features in this case are a set of colour regions or bins, and the dimensionality of the feature space depends on the size of the bins. An obvious advantage of this representation is that it is invariant with respect to many operations, and can compare images of different sizes. In [25] this is combined with the use of HSV (Hue Saturation Value) instead of RGB (Red Green Blue). The two representations are equivalent, but the first decorrelates the colour components (HS) from the luminance component (V), and is argued to be cognitively more plausible.

In order to specify a kernel however, one must also decide a measure of similarity between feature vectors. A general class of kernels can be written as follows:

$$K(\mathbf{x},\mathbf{z}) = \exp\left(-\frac{d(\mathbf{x},\mathbf{z})}{\sigma^2}\right)$$

where d is some measure of similarity between inputs. For histograms, a possible choice is the χ^2 function, approximated by

$$d(\mathbf{x},\mathbf{z}) = \sum_{i=1}^{n} \frac{(x_i - z_i)^2}{x_i + z_i}.$$

A second alternative is a simple p-norm

$$d_p(\mathbf{x},\mathbf{z}) = \left(\sum_{i=1}^{n} |x_i - z_i|^p\right)^{1/p}.$$

The kernel obtained using d_p for $p = 1, 2$ satisfies Mercer's conditions, although it is not known if this is true for the χ^2 choice, which nonetheless makes intuitive sense and performs well in practice. Other types of histograms could also be extracted from the images and used as feature vectors.

An image recognition task on images extracted from the Corel Stock Photo Collection was selected for this application. The Corel dataset contains about 200 categories, each with 100 images. In the experiments, a

subset denoted by Corel14 was selected comprising those pictures associated with the following fourteen categories: air shows, bears, elephants, tigers, Arabian horses, polar bears, African speciality animals, cheetahs leopards jaguars, bald eagles, mountains, fields, deserts, sunrises sunsets, and night-scenes. In this reduced dataset, there are 1400 photos. Each category was split into 2/3 for training and 1/3 for testing. The number of bins was fixed at 16 per colour dimension, so that the dimension of the histograms was $16^3 = 4096$. The kernels used are the ones described above, with the 1-norm, 2-norm and χ^2 used as similarity measures.

The results show almost a factor of 2 improvement over *k*-nearest neighbour, based on the same metric. Interestingly of the three metrics the performances using the 1-norm and χ^2 were very similar, while the 2-norm performed significantly worse in this case. Hence, for this problem the standard choice of a Gaussian kernel was not a good one.

Remark 8.4

Note that in both cases the number of examples was much lower than the feature space dimensionality, but that, despite this, good performance was achieved by maximising the margin. The possibility of using maximal margin machines also derives partly from the high dimensionality of the feature space.

Remark 8.5

A very important application in which SVMs have proven to be particularly effective is object detection in a visual scene. One example is face detection: given as input an arbitrary image, determine whether or not there are any human faces in the image, and if so where they are. The system developed by Osuna et al. [110], works by exhaustively scanning an image for face-like patterns at many possible scales, and uses an SVM as a classifier, to determine whether a given image is a human face. A database containing face/non-face patterns, represented as $19 \times 19 = 361$ pixel vectors, was used to train a soft margin classifier, with polynomial kernels of degree 2. A number of relevant technical details must be considered in this application, which cannot be reported here. An analogous problem involves detecting pedestrians in a visual scene for car-driving applications [108], where wavelets are used as feature extractors before being processed by an SVM with polynomial kernels. Both these applications are very important but they are too complex to describe in detail here. See Section 8.5 for pointers to the original papers.

8.3 Hand-written Digit Recognition

The first real-world task on which Support Vector Machines were tested was the problem of hand-written character recognition. This is a problem currently used for benchmarking classifiers, originally motivated by the need of the US Postal Service to automate sorting mail using the hand-written Zip codes. Different models of SVM have been tested on two databases of digits: USPS (United States Postal Service) and NIST (National Institute for Standard and Technology), both publicly available.

The USPS dataset comprises 7291 training and 2007 test patterns, represented as 256 dimensional vectors (16 × 16 matrices) with entries between 0 and 255. The NIST dataset, created for benchmarking purposes, contains 60000 training patterns and 10000 test patterns, the images are described as 20 × 20 matrices again with grey level entries.

This problem has been tackled by Vapnik and his co-workers [19], [27], [128], [78], both with maximal margin and with soft margin machines, mainly with polynomial and Gaussian kernels, though sigmoidal kernels have been used despite not satisfying Mercer's conditions. Furthermore, multi-class SVMs have been tested on these data. It is interesting not only to compare SVMs with other classifiers, but also to compare different SVMs amongst themselves. They turn out to have approximately the same performance, and furthermore to share most of their support vectors, independently of the chosen kernel. As expected, as the flexibility of the kernel map increases, the number of support vectors grows, but this happens quite slowly, and within a certain range of kernel parameter the performance appears to be robust. The results of experiments have been reported in a series of papers by Burges, Cortes, Schölkopf, Vapnik, etc., and summarised in [159].

For USPS data, where the input space is 256 dimensional, the following polynomial and Gaussian kernel were used:

$$K(\mathbf{x},\mathbf{y}) = \left(\frac{\langle \mathbf{x}\cdot\mathbf{y}\rangle}{256}\right)^d$$

and

$$K(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{256\sigma^2}\right)$$

for different values of d and .

For polynomial kernels, degrees from 1 to 6 have been tested, for Gaussian kernels, values of between 0.1 and 4.0. The kernel parameter certainly affects the generalisation. The USPS are reported to be totally separable with a maximal margin machine starting from degree 3, whereas with degree 1 the training error is 340/7291 and with degree 2 it is 4/7291, using 1-norm soft margin optimisation. The number of support vectors grows very slowly with the degree. For Gaussian kernels, one can use the adaptive kernels algorithm [31] in conjunction with a gradient ascent algorithm like the one described in Section 7.2, automatically scanning the space of possible kernel parameters.

This set of experiments is particularly interesting, because the data have been extensively studied and there are algorithms that have been designed specifically for this dataset, hence incorporating a great deal of prior knowledge about the problem. The fact that SVM can perform as well as these systems without including any detailed prior knowledge is certainly remarkable [78].

2 Chapter 8: Applications of Support Vector Machines

8.4 Bioinformatics

8.4.1 Protein Homology Detection

A protein is formed from a sequence of amino-acids from a set of 20, there exist thousands of different proteins, and one of the central problems of bioinformatics is to predict structural and functional features of a protein based on its amino-acidic sequence. This can be done by relating new protein sequences to proteins whose properties are already known, i.e. by detection of protein homologies. In a sense, proteins are clustered in families, which are then grouped together into super-families based on their properties.

Many techniques exist for detecting homologies between proteins, based on their sequence. One of the most common involves constructing a generative model of a protein family from positive examples, and using it to compute the degree of similarity of a new sequence with that family. The parameters of a statistical model, such as a Hidden Markov Model (HMM) $H(\ )$, are estimated using the examples in conjunction with general a priori information about properties of the proteins.

The model $H(\ )$ assigns a probability $P(x\&\#8739;H(\ ))$ to new protein sequence x, and new proteins from the same super-family are expected to have a higher score than those outside the family. The probability score is hence a measure of homology between a protein and a super-family. We will see that the generative model can also be exploited to produce a kernel. The aim of the experiments of Jaakkola and Haussler [65] was to see if an SVM built with such a kernel could effectively classify protein families into their super-family.

Kernel functions, however, should specify a similarity score between a pair of sequences, whereas the likelihood score from an HMM measures the closeness of the sequence to the model itself. There are, however, some intermediate quantities, computed by HMM systems during the ranking of a new query protein, that can be used to realise a mapping to a feature space, and hence also a kernel. They can be viewed as intermediate representations of the protein in the ‘internal language’ of the HMM. Without describing HMMs in detail, one of those quantities is the Fisher score, which for a sequence x is given by

$$U_x = \frac{\partial \log P(x|H(\boldsymbol{\theta}))}{\partial \boldsymbol{\theta}}$$

that is the gradient of the log-likelihood of the sequence x with respect to the parameters of the model $H(\ )$. The vector U_x can be viewed as an intermediate representation of the query sequence in the HMM: a representation that reflects the assumptions that went into building the HMM and hence also biological prior knowledge. It is closely related to the vector of sufficient statistics of the probability model that give a complete summary of the sequence in the internal language of the model and it is easily computed as a by-product of the evaluation of the HMM on a given sequence.

It is reasonable to expect that the distance between these vectors provides a measure of similarity between the corresponding sequences. A kernel can then be created using a Gaussian based on the 2-norm

$$K(x, y) = \exp\left(-\frac{\|U_x - U_y\|_2^2}{2\sigma^2}\right),$$

though it might be more natural to use the metric given by the Fisher information matrix. The kernel K is the inner product in the feature space that is the image of the composition of two feature maps

$$x \longmapsto U_x \longmapsto \phi(U_x),$$

where is the feature map associated with the Gaussian. Hence, this is certainly a Mercer kernel.

This system was implemented using a gradient ascent optimiser similar to the one described in Section 7.2. The positive training examples were a super-family excluding all members of a known family that then

formed the test set. Negative examples in both test and training sets came from other super-families.

The system significantly outperformed a number of state-of-the-art protein homology detection systems, proving particularly effective in the detection of remote homologies.

Remark 8.6

It is worth emphasising that again the kernel is constructed using domain knowledge encoded in the HMMs. Many other measures of distance between biological sequences have been proposed, including the use of edit distances, generative grammars, pair-HMMs, and so on. They all aim to exploit prior knowledge of the domain in the design of a similarity measure between inputs.

8.4.2 Gene Expression

Another application of SVMs to biological data mining is given by the automatic categorisation of gene expression data from DNA microarrays. The DNA microarray technology is revolutionising genetic analysis by making it possible to detect what genes are expressed in a particular tissue, and to compare the levels of expression of genes between the tissue in two different conditions. The levels of expression of thousands of genes can now easily be measured by biologists in a single experiment. As the data produced by DNA microarray experiments accumulate, it is becoming essential to have accurate means for extracting biological information and ultimately automatically assigning functions to genes. As an example one would like to use such data to determine whether a new gene encodes a protein of a certain class. Such a classifier could recognise new members of a protein class among genes of unknown function. This kind of classification task is very hard for standard pattern recognition systems as the datasets can be large, noisy and unbalanced, having many more negative than positive examples.

Brown et al. [21] describe a successful use of Support Vector Machines applied to gene expression data both for the task of classifying unseen genes and for the equally important one of cleaning existing datasets. We will not describe here the interesting experimental setup needed to produce microarray data. The experiments of Brown et al. used the following data for computer analysis. A total of 2467 genes of the budding yeast *S. cerevisiae* were represented by a 79 dimensional gene expression vector, and classified according to six functional classes. In most cases, the number of positives was less than 20 (less than 1%), and this, combined with the presence of noise, made the learning task particularly hard: a trivial solution in which all training data are classified as negative was usually found.

The SVM used was a variation of the 2-norm soft margin optimisation described in Chapter 6, where a diagonal element is added to the kernel matrix. In this case, in order to separately control the number of errors in the two classes, the diagonal element added to the kernel depended on the class. This was designed to produce smaller Lagrange multipliers in the dominant negative class, and larger multipliers in the small positive class. The ratio between the elements of the two classes was fixed to be roughly equal to the ratio between the sizes of the two classes. The kernels were chosen to be Gaussian distributions, whose was fixed to be roughly equal to the distance between the closest pair of examples of opposite class.

The results obtained were compared with Parzen windows, Fisher's linear discriminant and the decision tree algorithm C4.5. The SVM outperformed the other methods on all the datasets tested, in many cases by as much as a factor of 2. The system was then used to detect outliers and as a central engine of a data browser for biological data [22].

8.5 Further Reading and Advanced Topics

The applications described in this final chapter are just some examples of the many and diverse uses that have been made of Support Vector Machines in recent years. We have selected particular examples for inclusion in this book on the grounds that they are easily accessible or that they illustrate some important aspect of learning systems design. Many other important applications could not be described due to lack of space, or because of their complexity: these, together with more recent examples, can be accessed via the website [30].

One crucial design choice is deciding on a kernel. Creating good kernels often requires lateral thinking: many measures of similarity between inputs have been developed in different contexts, and understanding which of them can provide good kernels depends on insight into the application domain. Some applications described here use rather simple kernels, motivated by established information retrieval techniques, like the ones for computer vision [129], [17], [118], [25] or the ones for text categorisation [36], [67]. Still, even with such simple kernels and even if the number of features is much larger than the number of training examples, SVMs deliver a very accurate hypothesis, often outperforming other more standard algorithms. Other applications use very sophisticated kernels, like the ones for biosequences discussed in [65], where the kernel is provided by an entire hidden Markov model system. Both Watkins and Haussler have recently proposed elegant kernels for symbol-sequences (discussed in Section 3.7), and their application to problems like biosequence analysis and text categorisation could provide very interesting results.

In the biological data mining example [21], the kernel used was a simple Gaussian kernel, but the difficulty arose in the imbalance of the dataset, a problem that was solved with a technique similar to the 2-norm soft margin. The same technique was also used in another medical application: automatic diagnosis of TBC with Support Vector Machines [168] for separately controlling specificity and sensitivity of the hypothesis. The gene expression work also used another important feature of SVMs: the possibility of using them for data cleaning, as discussed in [55], where the fact that potential outliers can be sought among the support vectors is exploited. The data and experiment are available on the web-site [22].

Other computer vision applications include the works on face detection and pedestrian detection by Tomaso Poggio, Federico Girosi and their co-workers [108], [110], tuberculosis diagnosis [168], the object recognition experiments discussed in [129]. More applications of SVM can be found in the two edited books [132] and [149], and in the website of GMD-FIRST [53] and the others accessible via the website [30].

Tuning the parameters of the system is another important design issue. The heuristics used as an example in the introduction of this chapter for the choice of in Gaussian kernels is due to Jaakkola; in general, one can think of schemes that automatically adapt the parameters to the data, based on some learning principle. One such simple scheme is provided by [31].

These references are also given on the website http://www.support-vector.net, which will be kept up to date with new work, pointers to software and papers that are available on-line.

2 Chapter 8: Applications of Support Vector Machines

Appendix A: Pseudocode for the SMO Algorithm

```
target = desired output vector
point = training point matrix

procedure takeStep(i1, i2)
  if (i1 == i2) return 0
  alph1 = Lagrange multiplier for i1
  y1 = target[i1]
  E1 = SVM output on point[i1] - y1 (check in error cache)
  s = y1 * y2
  Compute L, H
  if (L == H)
    return 0
  k11 = kernel(point[i1] ,point[i1])
  k12 = kernel(point[i1] ,point[i2])
  k22 = kernel(point[i2] ,point[i2])
  eta = 2*k12-k11-k22
  if (eta < 0)
  {
    a2 = alph2 - y2*(E1-E2)/eta
    if (a2 < I) a2 = L
    else if (a2 > H) a2 = H
  }
  else
  {
    Lobj = objective function at a2=L
    Hobj = objective function at a2=H
    if (Lobj > Hobj+eps)
      a2 = L
    else if (Lobj < Hobj-eps)
      a2 = H
    else
      a2 = alph2
  }
  if (|a2-alph2| < eps*(a2+alph2+eps))
      return 0
    a1 = alph1+s*(alph2-a2)
    Update threshold to reflect change in Lagrange multipliers
    Update weight vector to reflect change in a1 & a2, if linear SVM
    Update error cache using new Lagrange multipliers
    Store a1 in the alpha array
    Store a2 in the alpha array
    return 1
  endprocedure

  procedure examineExample(i2)
    y2 = target [i2]
    alph2 = Lagrange multiplier for i2
    E2 = SVM output on point [i2] - y2 (check in error cache)
    r2 = E2*y2
    if ((r2 < -tol && alph2 < C) || (r2 > tol && alph2 > 0))
    {
      if (number of non-zero & non-C alpha > 1)
      {
        i1 = result of second choice heuristic
        if takeStep(i1,i2)
          return 1
      }
      loop over non-zero and non-C alpha, starting at random point
      {
          i1 = identity of current alpha
```

```
          if takeStep(i1, i2)
            return 1
      }
      loop over all possible i1, starting at a random point
      {
          i1 = loop variable
          if takeStep(i1, i2)
            return 1
      }
  }
  return 0
endprocedure

main routine:
  initialize alpha array to all zero
  initialize threshold to zero
  numChanged = 0
  examineAll = 1
  while (numChanged > 0 | examineAll)
    {
  numChanged = 0
  if (examineAll)
    loop I over all training examples
      numChanged += examineExample(I)
  else
    loop I over examples where alpha is not 0 & not C
      numChanged += examineExample(I)
  if (examineAll == 1)
    examineAll = 0
  else if (numChanged == 0)
    examineAll = 1
}
```

Appendix B: Background Mathematics

B.1 Vector Spaces

We begin with the definition of a vector space. Where appropriate, we will give simpler definitions, which at the expense of some generality will be sufficient for the use made of them in the text. For example a vector space can be defined over any field, but we will consider vector spaces over the real numbers, so that what we now introduce are sometimes called ‘real vector spaces’.

Definition B.1

A set X is a *vector space (VS)* if two operations (addition, and multiplication by scalar) are defined on X such that, for $\mathbf{x}, \mathbf{y} \in X$, and $\mathbb{R}$,

$$\begin{aligned} \mathbf{x}+\mathbf{y} &\in X, \\ \alpha\mathbf{x} &\in X, \\ 1\mathbf{x} &= \mathbf{x}, \\ 0\mathbf{x} &= \mathbf{0}, \end{aligned}$$

and such that in addition X is a commutative group with identity $\mathbf{0}$ under the addition operation and satisfies the distributive laws for scalar multiplication

$$\alpha(\mathbf{x}+\mathbf{y}) = \alpha\mathbf{x} + \alpha\mathbf{y},$$

and

$$(\alpha+\beta)\mathbf{x} = \alpha\mathbf{x} + \beta\mathbf{x},$$

The elements of X are also called *vectors* while the real numbers are referred to as *scalars*.

Example B.2

The standard example of a VS is the set $\mathbb{R}$″ of real column vectors of fixed dimension n. We use a dash to denote transposition of vectors (and matrices) so that a general column vector can be written as

$$\mathbf{x} = (x_1, \ldots, x_n)',$$

where $x_i \in \mathbb{R}$, $i = 1, \ldots, n$.

Definition B.3

A non-empty subset M of a vector space X is a *subspace* of X, if the restriction of the operations to M makes it a vector space.

Definition B.4

A *linear combination* of the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in a vector space is a sum of the form $\alpha_1\mathbf{x}_1 + \cdots + \alpha_n\mathbf{x}_n$ where $\alpha_i \in \mathbb{R}$. If the α_i are positive and $\sum_{i=1}^{n} \alpha_i = 1$, the sum is also called a *convex combination*.

It is easy to see that a linear combination of vectors from a subspace is still in the subspace, and that linear combinations can actually be used to build subspaces from an arbitrary subset of a VS. If S is a subset of X, we denote by span(S) the subspace of all possible linear combinations of vectors in S.

Definition B.5

A finite set of vectors $S = \{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$ is *linearly dependent* if it is possible to find constants $_1,\ldots, _n$ not all zero such that

$$\alpha_1\mathbf{x}_1 + \cdots + \alpha_n\mathbf{x}_n = \mathbf{0}.$$

If this is not possible, the vectors are called *linearly independent*.

This implies that a vector **y** in span(*S*), where *S* is a set of linearly independent vectors, has a unique expression of the form

$$\mathbf{y} = \alpha_1\mathbf{x}_1 + \ldots + \alpha_n\mathbf{x}_n,$$

for some *n* and $\mathbf{x}_i \quad S$, $i = 1,\ldots,n$.

Definition B.6

A set of vectors $S = \{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$ is said to form a basis for *X* if *S* is linearly independent, and if every $\mathbf{x} \quad X$ can be uniquely expressed as a linear combination of the vectors in the set *S*. Though there will always be many different bases, their sizes are always the same. The size of a basis of a vector space is known as its *dimension*.

Finite dimensional spaces are easier to study, as fewer pathological cases arise. Special requirements have to be added in order to extend to the infinite dimensional case some basic properties of finite dimensional VSs. We consider introducing a measure of distance in a VS.

Definition B.7

A *normed linear space* is a VS *X* together with a real-valued function that maps each element $x \quad X$ into a real number ||**x**|| called the norm of **x**, satisfying the following three properties:

1. **Positivity** $\|\mathbf{x}\| \geq 0, \forall \mathbf{x} \in X,$ equality holding if and only if $\mathbf{x} = \mathbf{0}$;
2. **Triangle inequality** $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|, \forall \mathbf{x}, \mathbf{y} \in X;$
3. **Homogeneity** $\|\alpha\mathbf{x}\| = |\alpha|\,\|\mathbf{x}\|, \forall \alpha \in \mathbb{R}, \text{ and } \forall \mathbf{x} \in X.$

Example B.8

Consider countable sequences of real numbers and let $1 \leq p <$ ∞. The space $_p$ is the set of sequences $\mathbf{z} = \{z_1,z_2,\ldots,z_i,\ldots\}$ such that

$$\|\mathbf{z}\|_p = \left(\sum_{i=1}^{\infty} |z_i|^p\right)^{1/p} < \infty.$$

The space $_{\∞}$ is formed of those sequences **z** that satisfy

$$\|\mathbf{z}\|_\infty = \max_{i\in\mathbb{N}} \left(|z_i|\right) < \infty.$$

The distance between two vectors **x** and **y** can be defined as the norm of their difference $d(\mathbf{x},\mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$.

Definition B.9

In a normed linear space an infinite sequence of vectors $\mathbf{x}_n$ is said to *converge* to a vector **x** if the sequence $\|\mathbf{x} - \mathbf{x}_n\|$ of real numbers converges to zero.

Definition B.10

A sequence $\mathbf{x}_n$ in a normed linear space is said to be a *Cauchy sequence* if ||$\mathbf{x}_n$ – $\mathbf{x}_m$|| 0 as *n,m* ∞. More precisely, given > 0, there is an integer N such that $\|\mathbf{x}_n - \mathbf{x}_m\| <$ for all $n,m > N$. A space is said to be *complete* when every Cauchy sequence converges to an element of the space.

Note that in a normed space every convergent sequence is a Cauchy sequence, but the converse in not always true. Spaces in which every Cauchy sequence has a limit are said to be *complete*. Complete normed linear spaces are called *Banach spaces*.

4 Appendix B: Background Mathematics

B.2 Inner Product Spaces

The theory of inner product spaces is a tool used in geometry, algebra, calculus, functional analysis and approximation theory. We use it at different levels throughout this book, and it is useful to summarise here the main results we need. Once again, we give the definitions and basic results only for the real case.

Definition B.11

A function f from a vector space X to a vector space Y is said to be *linear* if for all , β $\mathbb{R}$ and **x, y** X,

$$f(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha f(\mathbf{x}) + \beta f(\mathbf{y}).$$

Note that we can view the real numbers as a vector space of dimension 1. Hence, a real-valued function is linear if is satisfies the same definition.

Example B.12

Let $X = \mathbb{R}^n$ and $Y = \mathbb{R}^m$. A linear function from X to Y can be denoted by an $m \times n$ matrix $\mathbf{A}$ with entries $\mathbf{A}_{ij}$ so that the vector $\mathbf{x} = (x_1,\ldots,x_n)$ is mapped to the vector $\mathbf{y} = (y_1,\ldots,y_m)$ where

$$y_i = \sum_{j=1}^{n} \mathbf{A}_{ij} x_j,\ i = 1,\ldots,m.$$

A matrix with entries $\mathbf{A}_{ij} = 0$, for $i \neq j$, is called diagonal.

Definition B.13

A vector space X is called an *inner product space* if there is a bilinear map (linear in each argument) that for each two elements **x, y** X gives a real number denoted by $\mathbf{x} \cdot \mathbf{y}$ satisfying the following properties:

- $\mathbf{x} \cdot \mathbf{y} = \mathbf{y} \cdot \mathbf{x}$,
- $\mathbf{x} \cdot \mathbf{x} \geq 0$, and $\mathbf{x} \cdot \mathbf{x} = 0$ ⃖ $\mathbf{x} = \mathbf{0}$.

The quantity $\mathbf{x} \cdot \mathbf{y}$ is called the *inner product* of x and y, though it is also known as the dot product or scalar product.

Example B.14

Let $X = \mathbb{R}^n$, $\mathbf{x} = (x_1,\ldots,x_n)$, $\mathbf{y} = (y_1,\ldots,y_n)$. Let $_i$ be fixed positive numbers. The following defines a valid inner product:

$$\langle \mathbf{x} \cdot \mathbf{y} \rangle = \sum_{i=1}^{n} \lambda_i x_i y_i = \mathbf{x}' \Lambda \mathbf{y},$$

where is the $n \times n$ diagonal matrix with non zero entries $_{ii} = {}_i$.

Example B.15

Let $X = C[a,b]$ be the vector space of continuous functions on the interval $[a,b]$ of the real line with the obvious definitions of addition and scalar multiplication. For f, g X, define

$$\langle f \cdot g \rangle = \int_a^b f(t)g(t)dt.$$

From the definition of an inner product, two further properties follow:

- $\mathbf{0} \cdot \mathbf{y} = 0$,
- X is automatically a normed space, with the norm

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x} \cdot \mathbf{x} \rangle}.$$

Definition B.16

Two elements $\mathbf{x}$ and $\mathbf{y}$ of X are called *orthogonal* if $\mathbf{x} \cdot \mathbf{y} = 0$. A set $S = \{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$ of vectors from X is called *orthonormal* if $\mathbf{x}_i \cdot \mathbf{x}_j = {}_{ij}$, where ${}_{ij} = 1$ if $i = j$, and 0 otherwise. For an orthonormal set S, and a vector $\mathbf{y}$ X, the expression

$$\sum_{i=1}^{n} \langle \mathbf{x}_i \cdot \mathbf{y} \rangle \, \mathbf{x}_i$$

is said to be a *Fourier series* for $\mathbf{y}$.

If S forms an orthonormal basis each vector $\mathbf{y}$ is equal to its Fourier series.

Theorem B.17

(Schwarz inequality) In an inner product space

$$|\langle \mathbf{x} \cdot \mathbf{y} \rangle|^2 \leq \langle \mathbf{x} \cdot \mathbf{x} \rangle \, \langle \mathbf{y} \cdot \mathbf{y} \rangle$$

and the equality sign holds if and only if $\mathbf{x}$ and $\mathbf{y}$ are dependent.

Theorem B.18

For $\mathbf{x}$ and $\mathbf{y}$ vectors of an inner product space X

$$\begin{aligned} \|\mathbf{x}+\mathbf{y}\|^2 &= \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 + 2\langle \mathbf{x} \cdot \mathbf{y} \rangle, \\ \|\mathbf{x}-\mathbf{y}\|^2 &= \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\langle \mathbf{x} \cdot \mathbf{y} \rangle. \end{aligned}$$

Definition B.19

The *angle* between two vectors $\mathbf{x}$ and $\mathbf{y}$ of an inner product space is defined by

$$\cos\theta = \frac{\langle \mathbf{x} \cdot \mathbf{y} \rangle}{\|\mathbf{x}\| \, \|\mathbf{y}\|}.$$

If $|\mathbf{x} \cdot \mathbf{y}| = \|\mathbf{x}\| \, \|\mathbf{y}\|$, the cosine is 1, $= 0$, and $\mathbf{x}$ and $\mathbf{y}$ are said to be *parallel*. If $\mathbf{x} \cdot \mathbf{y} = 0$, the cosine is 0, $\theta = \frac{\pi}{2}$ and the vectors are said to be *orthogonal*.

Definition B.20

Given a set $S = \{\mathbf{x}_1,\ldots,\mathbf{x}^n\}$ of vectors from an inner product space X, the $n \times n$ matrix $\mathbf{G}$ with entries $\mathbf{G}_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j$ is called the *Gram matrix* of S.

B.3 Hilbert Spaces

Definition B.21

A space H is *separable* if there exists a countable subset $D \subset H$, such that every element of H is the limit of a sequence of elements of D. A Hilbert space is a complete separable inner product space.

Finite dimensional vector spaces such as $\mathbb{R}^n$ are Hilbert spaces.

Theorem B.22

Let H be a Hilbert space, M a closed subspace of H, and $\mathbf{x} \in H$. There is a unique vector $\mathbf{m}_0 \in M$, known as the projection of $\mathbf{x}$ onto M, such that

$$\|\mathbf{x} - \mathbf{m}_0\| \leq \inf \{\|\mathbf{x} - \mathbf{m}\| : \mathbf{m} \in M\}.$$

A necessary and sufficient condition for $\mathbf{m}_0 \in M$ to be the projection of $\mathbf{x}$ onto M is that the vector $\mathbf{x} - \mathbf{m}_0$ be orthogonal to vectors in M.

A consequence of this theorem is that the best approximation to $\mathbf{x}$ in the subspace M generated by the orthonormal vectors $\{\mathbf{e}_1,\ldots,\mathbf{e}_n\}$ is given by its Fourier series

$$\sum_{i=1}^{n} \langle \mathbf{x} \cdot \mathbf{e}_i \rangle \, \mathbf{e}_i.$$

This leads naturally to studying the properties of series like this in the case of infinite bases.

Definition B.23

If S is an orthonormal set in a Hilbert space H and no other orthonormal set contains S as a proper subset, that is S is maximal, then S is called an *orthonormal basis* (or: a *complete orthonormal system*) for H.

Theorem B.24

Every Hilbert space H has an orthonormal basis. Suppose that $S = \{\mathbf{x}_\alpha\}_{\alpha \in A}$ is an orthonormal basis for a Hilbert space H. Then, ∀$\mathbf{y} \in H$,

$$\mathbf{y} = \sum_{\alpha \in A} \langle \mathbf{y} \cdot \mathbf{x}_\alpha \rangle \, \mathbf{x}_\alpha$$

and $\|\mathbf{y}\|^2 = \sum_{\alpha \in A} |\mathbf{y} \cdot \mathbf{x}_\alpha|^2$.

This theorem states that, as in the finite dimensional case, every element of a Hilbert space can be expressed as a linear combination of (possibly infinite) basis elements.

The coefficients $\mathbf{y} \cdot \mathbf{x}_\alpha$ are often called the Fourier coefficients of $\mathbf{y}$ with respect to the basis $S = \{\mathbf{x}_\alpha\}_{\alpha \in A}$.

Example B.25

Consider countable sequences of real numbers. The Hilbert space ℓ_2 is the set of sequences $\mathbf{z} = \{z_1, z_2, \ldots, z_i, \ldots\}$ such that

$$\|\mathbf{z}\|_2^2 = \sum_{i=1}^{\infty} z_i^2 < \infty,$$

where the inner product of sequences **x** and **z** is defined by

$$\langle \mathbf{x} \cdot \mathbf{z} \rangle = \sum_{i=1}^{\infty} x_i z_i.$$

If $\boldsymbol{\mu} = \{\mu_1, \mu_2, \ldots, \mu_i, \ldots\}$ is a countable sequence of positive real numbers, the Hilbert space $_2(\boldsymbol{\mu})$ is the set of sequences $\mathbf{z} = \{z_1, z_2, \ldots, z_i, \ldots\}$ such that

$$\|\mathbf{z}\|_2^2 = \sum_{i=1}^{\infty} \mu_i z_i^2 < \infty,$$

where the inner product of sequences **x** and **z** is defined by

$$\langle \mathbf{x} \cdot \mathbf{z} \rangle = \sum_{i=1}^{\infty} \mu_i x_i z_i.$$

The normed space $_1$ is the set of sequences $\mathbf{z} = \{z_1, z_2, \ldots, z_i, \ldots\}$ for which

$$\|\mathbf{z}\|_1 = \sum_{i=1}^{\infty} |z_i| < \infty.$$

Example B.26

Consider the set of continuous real-valued functions on a subset X of $\mathbb{R}^n$. The Hilbert space $L_2(X)$ is the set of functions f for which

$$\|f\|_{L_2} = \int_X f(\mathbf{x})^2 d\mathbf{x} < \infty,$$

where the inner product of functions f and g is defined by

$$\langle f \cdot g \rangle = \int_X f(\mathbf{x}) g(\mathbf{x}) d\mathbf{x}.$$

The normed space $L_{\∞}(X)$ is the set of functions for which

$$\|f\|_{L_\infty} = \sup_{\mathbf{x} \in X} |f(\mathbf{x})| < \infty.$$

B.4 Operators, Eigenvalues and Eigenvectors

Definition B.27

A linear function from a Hilbert space H to itself is known as a *linear operator*. The linear operator A is *bounded* if there exists a number $\|A\|$ such that $\|Ax\| \leq \|A\| \, \|x\|$, for all $x \quad H$.

Definition B.28

Let A be a linear operator on a Hilbert space H. If there is a vector, $0 \neq \mathbf{x} \quad H$, such that $A\mathbf{x} = \quad \mathbf{x}$ for some scalar , then is an *eigenvalue* of A with corresponding *eigenvector* $\mathbf{x}$.

Definition B.29

A bounded linear operator A on a Hilbert space H is *self-adjoint* if

$$\langle A\mathbf{x} \cdot \mathbf{z} \rangle = \langle \mathbf{x} \cdot A\mathbf{z} \rangle ,$$

for all $\mathbf{x}, \mathbf{z} \quad H$. For the finite dimensional space $\mathbb{R}^n$ this implies that the corresponding $n \times n$ matrix $\mathbf{A}$ satisfies $\mathbf{A} = \mathbf{A}$, that is $\mathbf{A}_{ij} = \mathbf{A}_{ji}$. Such matrices are known as *symmetric*.

The following theorem involves a property known as compactness. We have omitted the definition of this property as it is quite involved and is not important for an understanding of the material contained in the book.

Theorem B.30

(Hilbert Schmidt) Let A be a self-adjoint compact linear operator on a Hilbert space H. Then there is a complete orthonormal basis $\{\boldsymbol{\phi}_i\}_{i=1}^{\infty}$ for H such that

$$A\boldsymbol{\phi}_i = \lambda_i \boldsymbol{\phi}_i,$$

and $_i \quad 0$ as $i \quad$ ∞.

In the finite case the theorem states that symmetric matrices have an orthonormal set of eigenvectors.

Definition B.31

A square symmetric matrix is said to be *positive (semi-) definite* if its eigenvalues are all positive (non-negative).

Theorem B.32

Let $\mathbf{A}$ be a symmetric matrix. Then $\mathbf{A}$ is positive (semi-) definite if and only if for any vector $\mathbf{x} \neq 0$

$$\mathbf{x}'\mathbf{A}\mathbf{x} > 0 \ (\geq 0).$$

Let $\mathbf{M}$ be any (possibly non-square) matrix and set $\mathbf{A} = \mathbf{M} \ \mathbf{M}$. Then $\mathbf{A}$ is a positive semi-definite matrix since we can write

$$\mathbf{x}'\mathbf{A}\mathbf{x} = \mathbf{x}'\mathbf{M}'\mathbf{M}\mathbf{x} = (\mathbf{M}\mathbf{x})'\,\mathbf{M}\mathbf{x} = \langle \mathbf{M}\mathbf{x} \cdot \mathbf{M}\mathbf{x} \rangle = \|\mathbf{M}\mathbf{x}\|^2 \geq 0,$$

for any vector $\mathbf{x}$. If we take $\mathbf{M}$ to be the matrix whose columns are the vectors $\mathbf{x}_i$, $i = 1,\ldots,n$, then $\mathbf{A}$ is the Gram matrix of the set $S = \{\mathbf{x}_1,\ldots,\mathbf{x}_n\}$, showing that Gram matrices are always positive semi-definite, and positive definite if the set S is linearly independent.

References

1. 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.
2. 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.
3. S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 1999. to appear.
4. J. K. Anlauf and M. Biehl. The adatron: an adaptive perceptron algorithm. *Europhysics Letters*, 10:687–692, 1989.
5. M. Anthony and P. Bartlett. *Learning in Neural Networks : Theoretical Foundations*. Cambridge University Press, 1999.
6. M. Anthony and N. Biggs. *Computational Learning Theory*, volume 30 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1992.
7. N. Aronszajn. Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68:337–404, 1950.
8. 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.
9. P. 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. MIT Press, 1999.
10. 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.
11. M. Bazaraa, D. Sherali, and C. Shetty. *Nonlinear Programming : Theory and Algorithms*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, 1992.
12. K. Bennett, N. Cristianini, J. Shawe-Taylor, and D. Wu. Enlarging the margin in perceptron decision trees. *Machine Learning*. to appear.
13. K. Bennett and A. Demiriz. Semi-supervised support vector machines. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems, 12*, pages 368–374. MIT Press, 1998.
14. K. P. Bennett and E. J. Bredensteiner. Geometry in learning. In C. Gorini, E. Hart, W. Meyer, and T. Phillips, editors, *Geometry at Work*. Mathematical Association of America, 1998.
15. K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.
16. C. M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, 1995.
17. 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. Springer Lecture Notes in Computer Science, Vol. 1112, 1996.
18. A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.
19. 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. ACM Press, 1992.
20. P. S. Bradley, O. L. Mangasarian, and D. R. Musicant. Optimization methods in massive datasets. Technical Report Data Mining Institute TR-99-01, University of Wisconsin in Madison, 1999.
21. M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. Furey, M. Ares, and D. Haussler. Knowledge-based analysis of microarray gene expression data using support vector machines. Technical report, University of California in Santa Cruz, 1999. (submitted for publication).
22. M. Brown, W. Grundy, D. Lin, N. Cristianini, C. Sugnet, T. Furey, M. Ares, and D. Haussler. Knowledge-based analysis of microarray gene expression data using support vector machines, 1999.

[http://www.cse.ucsc.edu/research/compbio/genex/genex.html]. Santa Cruz, University of California, Department of Computer Science and Engineering.
23. C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
24. C. Campbell and N.-Cristianini. Simple training algorithms for support vector machines. Technical Report CIG-TR-KA, University of Bristol, Engineering Mathematics, Computational Intelligence Group, 1999.
25. O. Chapelle, P. Haffner, and V. Vapnik. SVMs for histogram-based image classification. *IEEE Transaction on Neural Networks*, 1999.
26. C. Cortes. *Prediction of Generalization Ability in Learning Machines*. PhD thesis, Department of Computer Science, University of Rochester, USA, 1995.
27. C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.
28. T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, 14:326–334, 1965.
29. N. Cristianini, C. Campbell, and J. Shawe-Taylor. A multiplicative updating algorithm for training support vector machine. In *Proceedings of the 6th European Symposium on Artificial Neural Networks (ESANN)*, 1999.
30. N. Cristianini and J. Shawe-Taylor. An Introduction to Support Vector Machines: the web-site associated with the book, 2000.
31. N. Cristianini, J. Shawe-Taylor, and C. Campbell. Dynamically adapting kernels in support vector machines. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems, 11*. MIT Press, 1998.
32. 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*, pages 109–117. Morgan Kaufmann, 1998.
33. L. Devroye, L. Györfi, and G. Lugosi. *A Probabilistic Theory of Pattern Recognition*. Number 31 in Applications of mathematics. Springer, 1996.
34. R. Dietrich, M. Opper, and H. Sompolinsky. Statistical mechanics of support vector networks. *Physics Review Letters*, 82:2975, 1999.
35. R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.
36. S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *7th International Conference on Information and Knowledge Management*, 1998.
37. T. Evgeniou and M. Pontil. On the V dimension for regression in reproducing kernel Hilbert spaces. In *Algorithmic Learning Theory: ALT-99*. Springer-Verlag, 1999.
38. T. Evgeniou, M. Pontil, and T. Poggio. Regularization networks and support vector machines. In A.J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.
39. T. Evgeniou, M. Pontil, and T. Poggio. A unified framework for regularization networks and support vector machines. Technical Report CBCL Paper #171/AI Memo #1654, Massachusetts Institute of Technology, 1999.
40. R. Fisher. *Contributions to Mathematical Statistics*. Wiley, 1952.
41. R. Fletcher. *Practical methods of Optimization*. Wiley, 1988.
42. S. Floyd and M. Warmuth. Sample compression, learnability, and the Vapnik-Chervonenkis dimension. *Machine Learning*, 21(3):269–304, 1995.
43. 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*. Morgan Kaufmann, 1998.
44. T. Friess, N. Cristianini, and C. Campbell. The kernel-Adatron: a fast and simple training procedure for support vector machines. In J. Shavlik, editor, *Machine Learning: Proceedings of the Fifteenth International Conference*. Morgan Kaufmann, 1998.
45. T. Friess and R. Harrison. Support vector neural networks: the kernel adatron with bias and soft margin. Technical Report ACSE-TR-752, University of Sheffield, Department of ACSE, 1998.
46. T. Friess and R. Harrison. A kernel based adaline. In *Proceedings of the 6th European Symposium on*

Artificial Neural Networks (ESANN), 1999.
47. S. I. Gallant. Perceptron based learning algorithms. *IEEE Transactions on Neural Networks*, 1:179–191, 1990.
48. C. Gentile and M. K. Warmuth. Linear hinge loss and average margin. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems, 11*. MIT Press, 1999.
49. M. Gibbs and D. MacKay. Efficient implementation of Gaussian processes. Technical report, Department of Physics, Cavendish Laboratory, Cambridge University, UK, 1997.
50. M. N. Gibbs. *Bayesian Gaussian Processes for Regression and Classification*. PhD thesis, University of Cambridge, 1997.
51. F. Girosi. An equivalence between sparse approximation and support vector machines. *Neural Computation*, 10(6):1455–1480, 1998.
52. F. Girosi, M. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995.
53. GMD-FIRST. GMD-FIRST web site on Support Vector Machines. http://svm.first.gmd.de.
54. L. Gurvits. A note on a scale-sensitive dimension of linear bounded functionals in Banach spaces. In *Proceedings of Algorithmic Learning Theory, ALT-97*, 1997.
55. I. Guyon, N. Matić, and V. Vapnik. Discovering informative patterns and data cleaning. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smythand, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 181–203. MIT Press, 1996.
56. I. Guyon and D. Stork. Linear discriminant and support vector classifiers. In A.J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.
57. M. H. Hassoun. *Optical Threshold Gates and Logical Signal Processing*. PhD thesis, Department of ECE, Wayne State University, Detroit, USA, 1986.
58. D. Haussler. Convolution kernels on discrete structures. Technical Report UCSC-CRL-99-10, University of California in Santa Cruz, Computer Science Department, July 1999.
59. R. Herbrich, T. Graepel, and C. Campbell. Bayes point machines: Estimating the Bayes point in kernel space. In *Proceedings of IJCAI Workshop Support Vector Machines*, 1999.
60. R. Herbrich, K. Obermayer, and T. Graepel. Large margin rank boundaries for ordinal regression. In A.J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.
61. M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.
62. C. Hildreth. A quadratic programming procedure. *Naval Research Logistics Quarterly*, 4:79–85, 1957.
63. A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.
64. K. U. Höffgen, K. S. van Horn, and H. U. Simon. Robust trainability of single neurons. *Journal of Computer and System Sciences*, 50(1):114–125, 1995.
65. 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*. MIT Press, 1998.
66. T. S. Jaakkola and D. Haussler. Probabilistic kernel regression models. In *Proceedings of the 1999 Conference on AI and Statistics*, 1999.
67. T. Joachims. Text categorization with support vector machines. In *Proceedings of European Conference on Machine Learning (ECML)*, 1998.
68. 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. MIT Press, 1999.
69. W. Karush. *Minima of Functions of Several Variables with Inequalities as Side Constraints*. Department of Mathematics, University of Chicago, 1939. MSc Thesis.
70. 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. MIT Press, 1999.
71. M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994.

72. M. J. Kearns and R. E. Schapire. Efficient distribution-free learning of probabilistic concepts. *Journal of Computer and System Science*, 48(3):464–497, 1994. Earlier version appeared in FOCS90.
73. 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, Department of CSA, IISc, Bangalore, India, 1999. Technical Report No. TR-ISL-99-03.
74. S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. Technical report, Control Division, Department of Mechanical and Production Engineering, National University of Singapore, 1999. Technical Report Nc. CD-99-14.
75. J. Kivinen and M. K. Warmuth. Additive versus exponentiated gradient for linear prediction. *Information and Computation*, 132:1–64, 1997.
76. A. Kowalczyk. Maximal margin perceptron. In A.J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.
77. H. Kuhn and A. Tucker. Nonlinear programming. In *Proceedings of 2nd Berkeley Symposium on Mathematical Statistics and Probabilistics*, pages 481–492. University of California Press, 1951.
78. Y. LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Müller, E. Säckinger, P. Simard, and V. Vapnik. Comparison of learning algorithms for handwritten digit recognition. In F. Fogelman-Soulié and P. Gallinari, editors, *Proceedings ICANN'95 – International Conference on Artificial Neural Networks*, volume II, pages 53–60. EC2, 1995.
79. N. Littlestone and M. Warmuth. Relating data compression and learnability. Technical report, University of California, Santa Cruz, 1986.
80. D. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley, 1984.
81. D. MacKay. Introduction to Gaussian processes. In *Neural Networks and Machine Learning (NATO Asi Series)*; *Ed. by Chris Bishop*, 1999.
82. D. J. C. MacKay. A practical Bayesian framework for backprop networks. *Neural Computation*, 4:448–472, 1992.
83. O. Mangasarian. Generalized support vector machines. In A.J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.
84. O. L. Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Operations Research*, 13:444–452, 1965.
85. O. L. Mangasarian. Multi-surface method of pattern separation. *IEEE Transactions on Information Theory*, IT-14:801–807, 1968.
86. O. L. Mangasarian. *Nonlinear Programming*. SIAM, 1994.
87. O. L. Mangasarian. Mathematical programming in data mining. *Data Mining and Knowledge Discovery*, 42(1):183–201, 1997.
88. O. L. Mangasarian. Generalized support vector machines. Technical Report Mathematical Programming TR 98-14, University of Wisconsin in Madison, 1998.
89. O. L. Mangasarian and D. R. Musicant. Successive overrelaxation for support vector machines. Technical Report Mathematical Programming TR 98-18, University of Wisconsin in Madison, 1998.
90. O. L. Mangasarian and D. R. Musicant. Data discrimination via nonlinear generalized support vector machines. Technical Report Mathematical Programming TR 99-03, University of Wisconsin in Madison, 1999.
91. O. L. Mangasarian and D. R. Musicant. Massive support vector regression. Technical Report Data Mining Institute TR-99-02, University of Wisconsin in Madison, 1999.
92. MATLAB. *User's Guide*. The MathWorks, Inc., 1992.
93. David A. McAllester. Some PAC-Bayesian theorems. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pages 230–234. ACM Press, 1998.
94. David A. McAllester. PAC-Bayesian model averaging. In *Proceedings of the 12th Annual Conference on Computational Learning Theory*, pages 164–170. ACM Press, 1999.
95. 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.
96. 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.
97. C. A. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.
98. M.L. Minsky and S.A. Papert. *Perceptrons*. MIT Press, 1969. Expanded Edition 1990.
99. T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
100. J. J. More and S. J. Wright. *Optimization Software Guide*. Frontiers in Applied Mathematics, Volume 14. Society for Industrial and Applied Mathematics (SIAM), 1993.
101. B. A. Murtagh and M. A. Saunders. MINOS 5.4 user's guide. Technical Report SOL 83.20, Stanford University, 1993.
102. R. Neal. *Bayesian Learning in Neural Networks*. Springer Verlag, 1996.
103. R. M. Neal. Monte carlo implementation of Gaussian process models for Bayesian regression and classification. Technical Report TR 9702, Department of Statistics, University of Toronto, 1997.
104. A. B. Novikoff. On convergence proofs on perceptrons. In *Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622. Polytechnic Institute of Brooklyn, 1962.
105. 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, 1996.
106. M. Opper and F. Vivarelli. General bounds on Bayes errors for regression with Gaussian processes. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems, 11*. MIT Press. 1998.
107. M. Opper and O. Winther. Gaussian processes and SVM: Mean field and leave-one-out. In A.J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.
108. M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio. Pedestrian detection using wavelet templates. In *Proceedings Computer Vision and Pattern Recognition*, pages 193–199, 1997.
109. 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. IEEE, 1997.
110. E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of Computer Vision and Pattern Recognition*, pages 130–136, 1997.
111. E. Osuna and F. Girosi. Reducing run-time complexity in SVMs. In *Proceedings of the 14th International Conference on Pattern Recognition, Brisbane, Australia*, 1998. To appear.
112. 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. MIT Press, 1999.
113. J. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAGs for multiclass classification. In *Neural Information Processing Systems (NIPS 99)*, 1999. to appear.
114. J. C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, 1998.
115. T. Poggio. On optimal nonlinear associative recall. *Biological Cybernetics*, 19:201–209, 1975.
116. T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9), September 1990.
117. D. Pollard. *Convergence of Stochastic Processes*. Springer, 1984.
118. M. Pontil and A. Verri. Object recognition with support vector machines. *IEEE Trans. on PAMI*, 20:637–646, 1998.
119. K. Popper. *The Logic of Scientific Discovery*. Springer, 1934. First English Edition by Hutchinson, 1959.
120. C. Rasmussen. *Evaluation of Gaussian Processes and Other Methods for Non-Linear Regression*. PhD thesis, Department of Computer Science, University of Toronto. 1996. ftp://ftp.cs.toronto.edu/pub/carl/thesis.ps.gz.
121. R. Rifkin, M. Pontil, and A. Verri. A note on support vector machines degeneracy. Department of Mathematical Sciences CBCL Paper #177/A1 Memo #1661, Massachusetts Institute of Technology, June 1999.
122. F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1959.

123. S. Saitoh. *Theory of Reproducing Kernels and its Applications*. Longman Scientific & Technical, 1988.
124. A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 49:210–229, 1959.
125. C. Saunders, A. Gammermann, and V. Vovk. Ridge regression learning algorithm in dual variables. In J. Shavlik, editor, *Machine Learning: Proceedings of the Fifteenth International Conference*. Morgan Kaufmann, 1998.
126. 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/.
127. R. Schapire, Y. Freund, P. Bartlett, and W. Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, 1998.
128. 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, 1995.
129. B. Schölkopf. *Support Vector Learning*. R. Oldenbourg Verlag, 1997.
130. B. Schölkopf, P. Bartlett, A. Smola, and R. Williamson. Shrinking the tube: a new support vector regression algorithm. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems, 11*. MIT Press, 1998.
131. B. Schölkopf, P. Bartlett, A. Smola, and R. Williamson. Support vector regression with automatic accuracy control. In L. Niklasson, M. Bodén, and T. Ziemke, editors, *Proceedings of the 8th International Conference on Artificial Neural Networks*, Perspectives in Neural Computing, pages 147–152. Springer Verlag, 1998.
132. B. Schölkopf, C. J. C. Burges, and A. J. Smola. *Advances in Kernel Methods – Support Vector Learning*. MIT Press. 1999.
133. B. Schölkopf, J. Shawe-Taylor, A. Smola, and R. Williamson. Generalization bounds via the eigenvalues of the gram matrix. Technical Report NCTR-1999-035, NeuroCOLT Working Group, http://www.neurocolt.com, 1999.
134. B. Schölkopf, A. Smola, and K.-R. Müller. Kernel principal component analysis. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, editors, *Artificial Neural Networks – ICANN'97*. pages 583-588. Springer Lecture Notes in Computer Science, Volume 1327, 1997.
135. B. Schölkopf, A. Smola, R. Williamson, and P. Bartlett. New support vector algorithms. Technical Report NC-TR-98-031, NeuroCOLT Working Group, http://www.neurocolt.com, 1998.
136. B. Schölkopf, A. J. Smola, and K. Müller. Kernel principal component analysis. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 327–352. MIT Press, 1999.
137. J. Shawe-Taylor. Classification accuracy based on observed margin. *Algorithmica*, 22:157–172, 1998.
138. 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.
139. J. Shawe-Taylor and N. Cristianini. Robust bounds on generalization from the margin distribution. Technical Report NC-TR-1998-020, NeuroCOLT Working Group, http://www.neurocolt.com, 1998.
140. J. Shawe-Taylor and N. Cristianini. Further results on the margin distribution. In *Proceedings of the Conference on Computational Learning Theory, COLT 99*, pages 278–285, 1999.
141. J. Shawe-Taylor and N. Cristianini. Margin distribution and soft margin. In A.J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.
142. 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, 1999.
143. F. W. Smith. Pattern classifier design by linear programming. *IEEE Transactions on Computers*, C-17:367–372, 1968.

144. A. Smola and B. Schölkopf. On a kernel-based method for pattern recognition, regression, approximation and operator inversion. *Algorithmica*, 22:211–231, 1998.
145. A. Smola and B. Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 1998. Invited paper, in press.
146. A. Smola, B. Schölkopf, and K.-R. Müller. The connection between regularization operators and support vector kernels. *Neural Networks*, 11:637–649, 1998.
147. 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. University of Queensland, 1998.
148. A. J. Smola. *Learning with Kernels*. PhD thesis, Technische Universität Berlin, 1998.
149. A. J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans. *Advances in Large Margin Classifiers*. MIT Press, 1999.
150. P. Sollich. Learning curves for Gaussian processes. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems, 11*. MIT Press, 1998.
151. R. J. Solomonoff. A formal theory of inductive inference: Part 1. *Inform. Control*, 7:1–22, 1964.
152. R. J. Solomonoff. A formal theory of inductive inference: Part 2. *Inform. Control*, 7:224–254, 1964.
153. A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-posed Problems*. W. H. Winston, 1977.
154. A. M. Turing. Computing machinery and intelligence. *Mind*, 49:433–460, 1950.
155. L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, Nov 1984.
156. R. J. Vanderbei. LOQO user's manual – version 3.10. Technical Report SOR-97-08, Princeton University, Statistics and Operations Research, 1997. Code available at http://www.princeton.edu/rvdb/.
157. V. Vapnik. *Estimation of Dependences Based on Empirical Data [in Russian]*. Nauka, 1979. (English translation Springer Verlag, 1982).
158. V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag. 1995.
159. V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.
160. V. Vapnik and O. Chapelle. Bounds on error expectation for SVM. In A.J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.
161. V. Vapnik and A. Chervonenkis. A note on one class of perceptrons. *Automation and Remote Control*, 25, 1964.
162. 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.
163. V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition [in Russian]*. Nauka, 1974. (German Translation: W. Wapnik & A. Tscherwonenkis, *Theorie der Zeichenerkennung*, Akademie-Verlag. Berlin, 1979).
164. 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.
165. 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.
166. V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24, 1963.
167. V. Vapnik and S. Mukherjee. Support vector method for multivariant density estimation. In *Neural Information Processing Systems (NIPS 99)*. 1999. to appear.
168. K. Veropoulos, C. Campbell, and N. Cristianini. Controlling the sensitivity of support vector machines. In *Proceedings of IJCAI Workshop Support Vector Machines*, 1999.
169. M. Vidyasagar. *A Theory of Learning and Generalization*. Springer, 1997.
170. S. Vijayakumar and S. Wu. Sequential support vector classifiers and regression. In *Proceedings of the International Conference on Soft Computing (SOCO'99)*, pages 610–619, 1999.
171. G. Wahba. *Spline Models for Observational Data*, volume 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, 1990.

172. 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. MIT Press, 1999.
173. G. Wahba, Y. Lin, and H. Zhang. GACV for support vector machines. In A. J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.
174. C. Watkins. Dynamic alignment kernels. Technical Report CSD-TR-98-11, Royal Holloway, University of London, Computer Science department, January 1999.
175. C. Watkins. Dynamic alignment kernels. In A.J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.
176. C. Watkins. Kernels from matching operations. Technical Report CSD-TR-98-07, Royal Holloway, University of London, Computer Science Department, July 1999.
177. J. Weston and R. Herbrich. Adaptive margin support vector machines. In A.J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 1999.
178. J. Weston and C. Watkins. Support vector machines for multi-class pattern recognition. In *Proceedings of the 6th European Symposium on Artificial Neural Networks (ESANN)*, 1999.
179. B. Widrow and M. Hoff. Adaptive switching circuits. *IRE WESCON Convention record*, 4:96–104, 1960.
180. 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.

Index

A

Index

B

Index

C

Index

D

Index

E

Index

F

Index

G

Index

H

Index

I-J

Index

K

Index

L

Index

M

Index

N

Index

O

Index

P

Index

Q

Index

R

Index

S

Index

T

Index

U

Index

v

Index

W

List of Figures

Chapter 2: Linear Learning Machines

Chapter 3: Kernel-Induced Feature Spaces

Chapter 4: Generalisation Theory

Chapter 5: Optimisation Theory

Chapter 6: Support Vector Machines

List of Tables

Chapter 2: Linear Learning Machines

Chapter 7: Implementation Techniques

List of Examples

Chapter 2: Linear Learning Machines

Chapter 3: Kernel-Induced Feature Spaces

Chapter 4: Generalisation Theory

Chapter 5: Optimisation Theory

Chapter 6: Support Vector Machines

Chapter 7: Implementation Techniques

Chapter 8: Applications of Support Vector Machines

Appendix B: Background Mathematics

Back Cover

This is the first comprehensive introduction to Support Vector Machines (SVMs), a new generation learning system based on recent advances in statistical learning theory. SVMs deliver state-of-the-art performance in real-world applications such as text categorisation, hand-written character recognition, image classification, biosequences analysis, etc., and are now established as one of the standard tools for machine learning and data mining. Students will find the book both stimulating and accessible, while practitioners will be guided smoothly through the material required for a good grasp of the theory and its applications. The concepts are introduced gradually in accessible and self-contained stages, while the presentation is rigorous and thorough. Pointers to relevant literature and web sites containing software ensure that it forms an ideal starting point for further study. Equally, the book and its associated web site will guide practitioners to updated literature, new applications, and on-line software.