Aspects of Kernel Based Learning Algorithms

Ying Guo

B. Eng. Northwestern Polytechnical University
M. Eng. Northwestern Polytechnical University

November 2001

A thesis submitted for the degree of Doctor of Philosophy of The Australian National University

Department of Engineering
Faculty of Engineering and Information Technology
The Australian National University
Declaration

These doctoral studies were conducted with Professor Bob Williamson and Professor Peter Bartlett as supervisors, and Dr Jonathan Baxter as advisor. The contents of this thesis are the results of original research carried out by myself, in collaboration with others, and have not been submitted for a higher degree at any other university or institution.

Much of the work in this dissertation has been published or has been submitted for publication in refereed journals and conference proceedings. In some cases, the conference papers contain material overlapping with the journal publications. The following papers have been submitted to journals.


The following paper has been published in Conference Proceedings.


Ying Guo
November 19, 2001
Department of Engineering
Faculty of Engineering and Information Technology
The Australian National University
Acknowledgements

Firstly, I would like to thank my main supervisor, Professor Bob Williamson, for providing a motivating, enthusiastic, and critical atmosphere during the many discussions we had. It was a great pleasure to me to conduct this thesis under his supervision. I would also like to thank my second supervisor, Professor Peter Bartlett, whose help, stimulating suggestions and encouragement helped me in the time of research for this thesis. I also want to thank my advisor, Dr Jonathan Baxter for the input on this thesis.

There is one more unofficial supervisor for this thesis: Dr Alexander Smola. I thank him for the close collaboration on this thesis, as well as many useful discussions. There are many people to thank for the academic as well as friendly support. I want to thank Professor John Shawe-Taylor for the very useful discussion and his dedicated supervision. I want to thank Dr Philip Long for the opportunity to research at the National University of Singapore, and the very useful insights he offered for the research.

Discussions with other researchers, either on their visits to the ANU, or at a conference overseas, have helped to increase my understanding in the field of machine learning. In particular, I want to thank Shai Ben-David, Jyrki Kivinen, Wee Sun Lee, Llew Mason, Gunnar Rätsch, and Bernhard Schölkopf.

I would like to express my gratitude to all those who gave me the possibility to complete this thesis. I want to thank the staff and students in the Department of Engineering, Department of Telecommunications, and Computer science laboratory for helping me and making the PhD experience a pleasant one.

I also want to acknowledge the financial support provided by the ANU PhD Scholarship and International Postgraduate Research Scholarship which make this study possible.

Lastly, I would like to give my special thanks to my family, especially my husband, Weidong Su. His support and encouragement over the past three years enabled me to complete this work.
Abstract

This thesis studies the problem of supervised learning using kernel based methods. These methods map linear algorithms nonlinearly into a higher dimensional kernel-induced feature space and then solve a linear problem. A number of standard learning methods belong to this family, the most common one being Support Vector Machines. This thesis is concerned with three aspects of kernel based algorithms.

Firstly, instead of the generalization bound from the classical VC theory, we develop a simplified bound on the covering number of support vector machines. This bound shows a clear relationship between the kernel and the generalization performance of support vector machines. We then apply this new bound to an example, which shows that the bound can be considerably smaller than existing results that did not take account of the kernel. Furthermore, the new bound proves that a smoother kernel leads to a smaller bound on the covering number, which means that the capacity of the learning machine is reduced.

Secondly, we summarize the classical support vector classification and regression algorithms systematically. We then apply support vector algorithms for two nonlinear time series prediction problems. The experiments examine the effect of different cost functions and different kernels. For the mobile radio signal prediction, which is normally solved by linear methods, support vector algorithms also show good performance when nonlinear kernels are used.

In the last part of the thesis, we apply the kernel methods to voting methods and achieve excellent performance. We begin by describe a class of new algorithms, NormBoost. NormBoost minimizes a regularized risk functional where the regularization term is a function defined in an inner product space. We then prove a theoretical lower bound on the regularized risk functional, which makes an excellent stopping criterion for the algorithm. By defining the regularization term in different inner product spaces, we obtain different NormBoost algorithms. Like AdaBoost, NormBoost performs gradient descent so as to maximally reduce the regularized risk functional at each iteration. At the same time, due to the regularization term, NormBoost approx-
imates a smoother function than AdaBoost. That is, instead of only considering the example data, it also takes account of the capacity of the classifier. We show that NormBoost achieves better generalization performance than AdaBoost in noisy situations. Especially, we define the regularization term in a reproducing kernel Hilbert space (RKHS). A particular advantage of the RKHS based regularization is that the optimal linear combination of weak hypotheses can actually be written using only $m$ terms, where $m$ is the number of examples. Experimental results demonstrate that this algorithm outperforms AdaBoost with a marked improvement in convergence rate.
## Contents

Declaration i

Acknowledgements iii

Abstract v

1 Introduction 1

1.1 Machine learning ................................. 1

1.1.1 Artificial intelligence and machine learning ............ 1

1.1.2 Supervised learning ............................... 3

1.2 Generalization performance ........................ 6

1.2.1 VC Theory ........................................... 7

1.2.2 Covering numbers, margin and generalization performance ... 11

1.3 Maximal margin algorithms ........................ 16

1.3.1 Support vector machines .......................... 17

1.3.2 Boosting ............................................ 22

1.4 Outline of the thesis ................................... 24

2 Covering Numbers for Support Vector Machines 27

2.1 Introduction and previous results ...................... 27
2.2 Covering number bound .................................................. 36
  2.2.1 The optimal choice of $(a_s)_s$ and $j$ ......................... 36
  2.2.2 Main result .............................................................. 37
  2.2.3 Proof outline ......................................................... 38
2.3 Example ........................................................................ 39
2.4 Experiment ..................................................................... 42
2.5 Conclusions .................................................................. 44
2.6 Proof of Theorem 2.10 .................................................. 45
2.7 Proof that Lemma 2.14 cannot be improved .................... 58

3 Time Series Prediction Using Support Vector Machines 61
  3.1 Introduction .............................................................. 62
  3.2 Support vector algorithms ............................................ 63
    3.2.1 Basic idea of primal optimization problem ............... 64
    3.2.2 Support vector classification .................................. 64
    3.2.3 Support vector regression ...................................... 66
  3.3 Experiments ............................................................... 68
    3.3.1 The Mackey-Glass equation .................................. 69
    3.3.2 Radio signal ......................................................... 73
  3.4 Discussion ................................................................. 78

4 Norm-Based Regularization of Boosting 81
  4.1 Introduction .............................................................. 81
  4.2 Basic idea ................................................................. 83
    4.2.1 The cost function ................................................. 83
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2.2</td>
<td>A Hilbert space on $X^m$</td>
<td>84</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Reproducing kernel Hilbert spaces</td>
<td>85</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Representer Theorem</td>
<td>86</td>
</tr>
<tr>
<td>4.3</td>
<td>Gradient descent</td>
<td>87</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Basic idea</td>
<td>87</td>
</tr>
<tr>
<td>4.3.2</td>
<td>A lower bound on $R_{\text{reg}}[f, S^m]$</td>
<td>89</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Gradient descent in function space</td>
<td>91</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Gradient descent in coefficient space</td>
<td>93</td>
</tr>
<tr>
<td>4.4</td>
<td>Experiments</td>
<td>95</td>
</tr>
<tr>
<td>4.4.1</td>
<td>$\ell_2^X$ NormBoost and AdaBoost</td>
<td>97</td>
</tr>
<tr>
<td>4.4.2</td>
<td>RKHS NormBoost and AdaBoost</td>
<td>99</td>
</tr>
<tr>
<td>4.5</td>
<td>Discussion</td>
<td>103</td>
</tr>
<tr>
<td>4.6</td>
<td>Generalization performance</td>
<td>106</td>
</tr>
<tr>
<td>5</td>
<td>Discussion and Conclusions</td>
<td>111</td>
</tr>
<tr>
<td>5.1</td>
<td>Summary</td>
<td>111</td>
</tr>
<tr>
<td>5.2</td>
<td>Further work</td>
<td>112</td>
</tr>
</tbody>
</table>

Bibliography | 114
List of Figures

1.1 Structural risk minimization ................................................................. 10
1.2 Operator norm. ..................................................................................... 12
1.3 The $\epsilon$-covering number. ............................................................... 14
1.4 Maximal margin classification ............................................................... 17
1.5 A binary classification problem. .......................................................... 19

2.1 Bounds of entropy numbers for SV machines (1). ............................ 33
2.2 Bounds of entropy numbers for SV machines (2). ............................ 33
2.3 Entropy number versus covering number. ........................................... 41
2.4 "Effective dimension" versus covering number. ................................. 42
2.5 "Effective dimension" versus entropy number. .................................... 43
2.6 Test error versus log of the kernel width. ........................................... 44

3.1 The $\epsilon$-insensitive band for regression problem. ............................ 65
3.2 The cost functions for regression problem. .......................................... 67
3.3 Test error versus delay $\tau$ for support vector regression. .................... 69
3.4 Mackey-Glass time series and $\tau = 17$ step ahead prediction. ............ 70
3.5 Test error for $\tau = 17$ step ahead prediction. ...................................... 71
3.6 Test error versus delay for two different cost functions. ...................... 74
3.7 The complex time series signal. .................................................. 75
3.8 Test error versus delay $\tau$ for classification. .......................... 76
3.9 Comparison of the test error for SVC and SVR. ......................... 77
3.10 Compare SVR with AR ............................................................. 78

4.1 Test error versus $\lambda$ for $\ell_2^X$ NormBoost Algorithm. ........ 96
4.2 The margin distribution for AdaBoost and NormBoost ............ 97
4.3 $R_{reg}$ versus the iteration time $t$ for $\ell_2^X$ NormBoost Algorithm. 98
4.4 Test error versus $\lambda$ for RKHS NormBoost Algorithm (1). .... 99
4.5 Test error versus $\lambda$ for RKHS NormBoost Algorithm (2). ... 100
4.6 $R_{reg}$ versus the iteration time $t$ for RKHS NormBoost Algorithm. 101
4.7 Test error versus $\lambda$. ............................................................ 102
4.8 $R_{reg}$ versus the iteration time $t + 1$ for RKHS NormBoost Algorithm. 103
4.9 Test error versus $\lambda$. ............................................................ 104
4.10 $R_{reg}$ versus the iteration time $t + 1$ for RKHS NormBoost algorithm for BANANA dataset. .................................................. 105
4.11 $R_{reg}$ versus the iteration time $t$ (1). ..................................... 106
4.12 $R_{reg}$ versus the iteration time $t$ (2). ..................................... 107
Notation

(a_s)_s \quad \text{sequence } (a_1, a_2, \cdots)

c(y, f(x)) \quad \text{cost function at } (x, y) \text{ for the output } f(x)

C \quad \text{regularization constant } (C = 1/\lambda)

C_k \quad \text{kernel constant}

\mathbb{C} \quad \text{the set of complex numbers}

d \quad \text{dimensionality of input space}

e_n(X) \quad n\text{-th dyadic entropy number of a set } X

\mathcal{E} \quad \text{ellipsoid containing } \Phi(\mathcal{X})

\mathcal{F} \quad \text{hypothesis class}

\mathcal{F}|_S \quad \text{restriction of } \mathcal{F} \text{ to the set } S

f \quad \text{hypothesis}

F[f(x)](\omega) \quad \text{Fourier transform of } f(x)

\text{fat}_{\mathcal{F}} \quad \text{fat shattering dimension of } \mathcal{F}

\mathcal{H} \quad \text{Hilbert space}

\mathcal{H}_k \quad \text{reproducing kernel Hilbert space (RKHS)}

i \quad \text{the imaginary unit}

k \quad \text{kernel}

K \quad \text{kernel matrix}

K_{ij} \quad \text{matrix entries } k(x_i, x_j)

\mathcal{L}(E, F) \quad \text{space of linear operators between Banach spaces } E, F

m \quad \text{length of sample}

\mathcal{N} \quad \text{covering number of a set / class of functions}
\( \mathbb{N} \) the set of natural numbers

\( \mathbb{N} \) embedding dimension

\( P(x, y) \) probability distribution on \( \mathcal{X} \times \mathcal{Y} \)

\( \Pi_\mathcal{F} \) growth function of class \( \mathcal{F} \)

\( RP[f] \) expected risk of \( f \) with respect to probability distribution \( P \)

\( R_{\text{emp}}[f, S^m] \) empirical risk of \( f \) on the basis of \( S^m \)

\( R_{\text{reg}}[f, S^m] \) regularized risk of \( f \) on the basis of \( S^m \)

\( R_{\text{emp}}'[f, S^m] \) empirical margin risk of \( f \) on the basis of \( S^m \)

\( \mathbb{R} \) the set of real numbers

\( \mathcal{S} \) feature space

\( S^m \) training set, \( S^m = (X^m, Y^m) \), sometimes also \( S \)

\( U_r(x) \) ball with radius \( r \) around \( x \)

\( \text{VCdim}(\mathcal{F}) \) VC dimension of \( \mathcal{F} \)

\( w \) weight vector

\( \mathcal{X} \) input space

\( X^m \) set of input values, \( X^m = (x_1, \ldots, x_m) \)

\( x_i \) the \( i \)-th pattern of input (\( x_i \in S^m \subset \mathcal{X} \))

\( \mathcal{Y} \) output space

\( Y^m \) set of target values, \( Y^m = (y_1, \ldots, y_m) \)

\( y_i \) the \( i \)-th pattern of output (\( y_i \in Y \subset \mathcal{Y} \))

\( \alpha_i \) Lagrange multiplier, also weight in boosting

\( \alpha \) vector of all Lagrange multipliers, also weight vector in boosting

\( \beta_i \) derivative of cost function

\( \beta \) vector of all \( \beta_i \)

\( \gamma \) margin, (sometimes also \( \epsilon \))

\( \delta \) variable, normally \( \delta > 0 \)

\( \delta_{ij} \) Kronecker delta

\( \epsilon_n(X) \) \( n \)-th entropy number of a set \( X \)
\begin{align*}
\eta_i & \quad \text{Lagrange multiplier} \\
\eta & \quad \text{vector of all Lagrange multipliers} \\
\kappa & \quad \text{constant} \\
\lambda & \quad \text{regularization constants (} \lambda = 1/C \text{), also eigenvalue of operator} \\
\xi_i & \quad \text{slack variables} \\
\sigma & \quad \text{variance, also kernel width} \\
\tau & \quad \text{delay in a time series} \\
\Phi & \quad \text{map into feature space} \\
\phi_i & \quad \text{basis functions for feature map} \\
\psi_j & \quad \text{eigenfunction of operator} \\
\omega & \quad \text{frequency} \\
\Omega[f] & \quad \text{regularizer} \\
\nabla & \quad \text{gradient operator} \\
\langle x, y \rangle & \quad \text{inner product between patterns } x \text{ and } y \\
|S| & \quad \text{cardinality of a finite set } S \\
\| \cdot \| & \quad \text{norm (normally Euclidean norm } \| x \| := \sqrt{\langle x, x \rangle} \text{)} \\
\| \cdot \|_p & \quad \text{p-norm, e.g. } (\sum_i |x_i|^p)^{1/p} \\
\| A \| & \quad \text{operator norm of } A \text{, i.e. sup}_x \| A x \| / \| x \| \\
\ell_p^d & \quad \text{sequence space} \\
L_p(\mathcal{X}, \mathbb{K}) & \quad \text{function space of real valued functions } f : \mathcal{X} \to \mathbb{K} \\
\mathbf{I} & \quad \text{identity operator, also unit matrix}
\end{align*}
Chapter 1

Introduction

1.1 Machine learning

1.1.1 Artificial intelligence and machine learning

Artificial Intelligence (AI) is concerned essentially with devising computer programs to make computers smarter. Research in AI is focused on developing computational approaches to intelligent behavior. This research has several goals, such as making machines and computational processes more useful, and understanding intelligence.

An exciting and potentially far-reaching development in contemporary computer science is the invention and application of methods of Machine Learning (ML). These methods enable a computer program to automatically analyze a large body of data and decide what information is most relevant. This crystallized information can then be used to help people make decisions faster and more accurately. ML is typically thought of as a subtopic of AI. It is a combination of the research in several areas such as Pattern Recognition, Regression Estimation, Computer Science, Statistics and Cognitive Sciences.

Although we are still far from machines that can learn like we do, much progress has been made in the past few decades. Research on machine learning started with the seminal work by A. Samuel [34], who developed a learning checker playing program
during the 1950s. The system learned by replaying games among grandmasters and by playing against itself. After considerable training, it performed quite well (certainly given the computers available in those days). Today, the same type of technique is incorporated in the TD-gammon system by G. Tesauro [46] that plays at the world-champion level, and it is also being applied to more complex games such as chess. *Machine learning* has been applied to very broad areas, such as the following:

1. **Bioinformatics** In biological research, there is a large and increasing amount of data available. The analysis of these biological sequence data is very important. For example, it can help people to determine the structure or the model of biological molecules. Machine learning has provided powerful techniques for modeling, searching and annotating this data. As the demands and opportunities for interpreting these data increase, the application of machine learning continues to expand.

2. **Computer vision** Computer vision is concerned with enabling a computer to see – to identify or understand what it sees, to locate what it is looking for, and so on. Computer vision can be used for face recognition, image recognition and in many other areas, and is expected to be a learning machine, which can adjust to a new environment or tasks. Machine learning can offer effective methods for automating the acquisition of visual models, adapting task parameters and representation, transforming signals to symbols, building trainable image processing systems, or focusing attention on target objects.

3. **Natural language processing** Natural language processing is concerned with computer-based understanding of natural language, text understanding, and related applications. So it is another ideal application area for machine learning. Machine learning can provide natural language processing with a range of alternative learning algorithms as well as a variety of general ideas. There are many successful applications of machine learning in natural language processing, such as speech recognition, information extraction, and machine translation.
1.1.2 Supervised learning

Based on the learning tasks, machine learning can be classified as supervised learning or unsupervised learning. *Supervised learning* is a type of machine learning where the learning algorithm is provided with a set of inputs for the algorithm along with the corresponding correct outputs, and learning involves the algorithm comparing its current actual output with the correct or target outputs, so that it knows what its error is, and modifies things accordingly. *Unsupervised learning* signifies a mode of machine learning where the system is not told the “right answer”. For example, it is not trained on pairs consisting of an input and the desired output, instead the system is given the input patterns and is left to find interesting patterns, regularities, or clusterings among them. This thesis will only focus on *supervised learning* algorithms.

In order to be able to apply quantitative analysis, we will formalize the supervised learning problem and make the standard assumptions about supervised learning for pattern recognition and regression estimation.

In general, for a supervised learning system, there are some patterns which have desired responses that are known (i.e., the concentration of target determinants). These two types of data, the representation of the objects and their responses in the system, form pairs which for the present purpose are called *inputs* $x$ from the input space $\mathcal{X}$ and *outputs* $y$ from the output space $\mathcal{Y}$. The system which returns an output value to every input vector is called a *supervisor*.

For a general learning algorithm, a sequence of observations, $(x_1, y_1), \ldots, (x_m, y_m)$ of patterns $x_i \in \mathcal{X}$ and target values $y_i \in \mathcal{Y}$ for $i = 1, \ldots, m$, is provided independently and identically distributed (iid) according to a fixed, but unknown probability distribution $P(x, y)$ on $\mathcal{X} \times \mathcal{Y}$. The input space $\mathcal{X}$ depends upon the number and domain of input variables. Typically, $\mathcal{X}$ is taken to be some subset of $\mathbb{R}^d$ and $\mathcal{Y}$ some subset of $\mathbb{R}$:

$$(x_1, y_1), \ldots, (x_m, y_m) \in \mathbb{R}^d \times \mathbb{R}. \quad (1.1)$$

We will use the shorthand $X^m = (x_1, \ldots, x_m)$, $Y^m = (y_1, \ldots, y_m)$, and $S^m = (X^m, Y^m)$. The sequence $S^m$ (sometimes also $S$) is generally called a *training set*,
which is assumed to be distributed according to the product probability distribution $P^m(x, y)$.

Our goal is to use the information contained in $S^m$ to find a function $f : \mathcal{X} \to \mathcal{Y}$ from a function class $\mathcal{F}$ such that the value of $f$ corresponds with the most likely label $y_i$ at location $x_i$. We hope that such a function $f$ can be used to estimate the unseen examples given only their inputs. We call $f$ the *decision function* or the *hypothesis*, and $\mathcal{F}$ the *hypothesis class*.

In order to choose the best approximation, one measures the cost $c(y, f(x))$ between the response $y$ of the supervisor to a given input $x$ and the corresponding hypothesis $f(x)$ provided by the learning machine. Consider the expected value of the cost, given by the expected risk functional

$$R_P[f] = \int c(y, f(x)) dP(x, y).$$

The goal is to find the *hypothesis* $f$ which minimizes $R_P[f]$. Therefore, the probability that $f$ incorrectly estimates a new example is small. As stated before, the joint probability distribution $P(x, y)$ is unknown, and the only information one has is contained in the training set $S^m$.

In the following, we denote the logarithms to base 2 and $e$ by $\log$ and $\ln$ respectively. We denote the input dimensionality and the number of examples by $d$ and $m$ respectively.

**Pattern recognition model**

For binary pattern recognition, the output space is defined as $\mathcal{Y} = \{-1, 1\}$, and the hypothesis class $\mathcal{F}$ is a set of indicator functions $f$ which only take the value $-1$ and $+1$. We call $f$ a classifier. The cost function is defined as $c : \{-1, 1\} \times \mathbb{R} \to \mathbb{R}$ for pattern recognition. Multi-class pattern recognition problems (where one has more than two classes) can be solved using voting methods based on combining many binary classification decision functions (see, e.g., [38]). Thus the technical results we obtained in the binary case can be easily extended to the multi-class case.
Consider the following cost function
\[
c(y_i, f(x_i)) = \begin{cases} 
-1 & \text{if } y_i \neq f(x_i) \\
1 & \text{if } y_i = f(x_i).
\end{cases}
\] (1.3)

The pattern recognition problem is to find the hypothesis \( f \) which minimizes the expected risk functional (1.2) with the cost function (1.3) when \( P(x, y) \) is unknown, but \( S^m \) is given.

**Regression model**

For the regression case, the supervisor's output \( y \) is a real value, and the hypothesis class \( \mathcal{F} \) is a set of real functions \( y \in \mathbb{R} \). The cost function is defined as \( c : \mathbb{R} \times \mathbb{R} \to \mathbb{R} \) for regression.

Consider the following cost function
\[
c(y_i, f(x_i)) = (y_i - f(x_i))^2.
\] (1.4)

The regression problem is to find the hypothesis \( f \) which minimizes the expected risk functional (1.2) with the cost function (1.4) when \( P(x, y) \) is unknown, but \( S^m \) is given. There are many other cost functions one could use. Some more examples are given in Subsection 4.2.1.

**The empirical risk functional**

Since we do not know the probability distribution \( P(x, y) \), it is necessary to design the approximation of \( R_P[f] \) from the training set \( S^m \). In practice, such approximation is usually measured in terms of a so-called *empirical risk functional*

\[
R_{\text{emp}}[f, S^m] := \frac{1}{m} \sum_{i=1}^{m} c(y_i, f(x_i)),
\] (1.5)

which is constructed on the basis of the training set \( S^m \).

The empirical risk functional is defined on a single hypothesis \( f \). However the goal of learning algorithms is to pick the best \( f \) from the hypothesis class \( \mathcal{F} \), given
the training sample $S^m$. One way of doing so is the Empirical Risk Minimization (ERM) inductive principle [51], which selects the learning hypothesis $f^* \in \mathcal{F}$ that minimizes the empirical risk

$$f^* := \arg\min_{f \in \mathcal{F}} R_{\text{emp}}[f, S^m].$$

In learning theory the ERM principle plays a crucial role. With different cost functions, one can apply ERM to specific learning problems. For example, we could define

$$R_{\text{emp}}[f, S^m] = \frac{1}{m} \sum_{i=1}^{m} (y_i - f(x_i))^2. \tag{1.6}$$

Minimization of (1.6) over $f$ is the classical least squares method.

1.2 Generalization performance

Once the cost function $c(y, f(x))$ is fixed, the empirical risk $R_{\text{emp}}[f, S^m]$ can easily be determined from the training set $S^m$ and it provides a simple estimate of the "true risk", the expected risk functional $R_P[f]$. It is to be hoped that, after training, the probability of a new example being generated according to $P(x, y)$ which cannot be correctly estimated by $f$ is small. This characteristic is known as good generalization performance of $f$ [51, 2].

The following questions that relate to generalization performance are fundamental in machine learning:

- What governs the generalization performance of a hypothesis?
- How can one control the generalization performance of a hypothesis?
- How can one construct algorithms that can control the generalization performance?

The theory that attempts to answer these questions is known as statistical learning theory [51, 52]. The tools used in statistical learning theory can be applied to the development of new algorithms which exhibit significantly better performance.
In the first part of this section we will give some technical results which relate to the generalization performance of functions. Then we will see how these results can be applied to quantify the generalization performance of the special algorithms we are interested in.

1.2.1 VC Theory

By treating the probability of an error on a particular sample as a binomial random variable and bounding the tail of this distribution, the probability of an error for a finite hypothesis class, \(|\mathcal{F}| < \infty\), can be bounded. This procedure leads to the following theorem [49].

**Theorem 1.1 (Upper bound for finite function class)** If \( f \in \mathcal{F} \) minimizes the empirical risk on a training set \( S \) with \( m \) samples, then with probability at least \( 1 - \delta \)

\[
R_P[f] \leq R_{\text{emp}}[f, S^m] + \sqrt{\frac{2}{m} \ln \left( \frac{2|\mathcal{F}|}{\delta} \right)}.
\]

Theorem 1.1 shows that the learning problem can be solved if the class \( \mathcal{F} \) of functions is finite. However, the requirement that \(|\mathcal{F}| < \infty\) is very restrictive in practice. In fact, even a class of functions parameterized by one real number fails to meet this condition. Next we show two measures of the complexity of a function class, the growth function and the Vapnik-Chervonenkis dimension (VC-dimension).

**Definition 1.2 (Growth function)** Assume \( S \) is a finite subset of the input space \( \mathcal{X} \). For a function class \( \mathcal{F} \), the restriction of \( \mathcal{F} \) to the set \( S \) is denoted by \( \mathcal{F}_S \). The growth function \( \Pi_{\mathcal{F}}(m) \) of \( \mathcal{F} \) of functions \( f : \mathcal{X} \to \{-1, +1\} \) is defined as

\[
\Pi_{\mathcal{F}}(m) := \max\{|\mathcal{F}_S| : S \subseteq \mathcal{X}, |S| = m\}.
\]

Here the cardinality of a finite set \( S \) is denoted by \(|S|\).
CHAPTER 1. INTRODUCTION

This definition shows that for a hypothesis class $\mathcal{F}$ of classifiers, the growth function of $\mathcal{F}$ measures the maximum possible number of unique classifications of a set of examples of size $m$ that can be made using classifiers from $\mathcal{F}$. Clearly, for all $m$, $\Pi_{\mathcal{F}}(m) \leq |\mathcal{F}|$ if $\mathcal{F}$ is finite and $\Pi_{\mathcal{F}}(m) \leq 2^m$. If classifiers from $\mathcal{F}$ can produce all possible classifications of an example set $S \subseteq \mathcal{X}$ (i.e., $|\mathcal{F}|_S = 2^{|S|}$), then we say that $\mathcal{F}$ shatters $S$. An important feature of shattering is that every possible classification of the points must be produced.

**Definition 1.3 (VC dimension)** Assume $S$ is a finite subset of the input space $\mathcal{X}$. The VC dimension of a hypothesis class $\mathcal{F}$ of classifiers $f : \mathcal{X} \to \{-1, 1\}$ is defined by

$$VCDim(\mathcal{F}) := \max \{|S| : S \subseteq \mathcal{X} \text{ and } \mathcal{F} \text{ shatters } S\}$$

or $\infty$ if no maximum exists.

Note that the definition of VC dimension only requires that there be one set of points that can be shattered, but not that all sets of points can be shattered.

The growth function $\Pi_{\mathcal{F}}(m)$ is a measure of how many different classifications of an $m$-sample into positive and negative examples can be achieved by the hypothesis $f$, while the VC dimension of $\mathcal{F}$ is the maximum value of $m$ for which $\Pi_{\mathcal{F}}(m) = 2^m$. In other words, the size of the largest set of examples which can be arbitrarily classified using classifiers from $\mathcal{F}$. Clearly, if $\mathcal{F}$ is finite then $VCDim(\mathcal{F}) \leq \log_2(|\mathcal{F}|)$.

Since the VC dimension can be defined in terms of the growth function, these two quantities are clearly related. The following result gives another relationship between the growth function and the VC dimension. This result, independently discovered by Vapnik and Chervonenkis [53], Sauer [35], and Shelah [43], is generally referred to as Sauer's Lemma.

**Lemma 1.4 (Sauer's lemma)** Let $d \geq 0$ and $m \geq 1$ be given integers and let $\mathcal{F}$ be a hypothesis space with $VCDim(\mathcal{F}) = d$, then

$$\Pi_{\mathcal{F}}(m) \leq \sum_{i=0}^{d} \binom{m}{i} \left< \left( \frac{em}{d} \right) \right>^i$$
where the second inequality holds for \( m \geq d \). Here \( e \) is the base of natural logarithm.

**Theorem 1.5** (Vapnik and Chervonenkis [53]) There exists a constant \( \kappa \) such that with probability at least \( 1 - \delta \) over the random choice of \( S^m \) according to \( \mathbb{P}^m(x, y) \), every \( f \in \mathcal{F} \) satisfies

\[
R_P[f] \leq R_{\text{emp}}[f, S^m] + \kappa \sqrt{\frac{\ln \Pi_\mathcal{F}(2m) + \ln(1/\delta)}{m}}.
\]

If we apply Sauer's Lemma to bound the growth function in terms of the VC dimension, we immediately obtain the following result, involving the VC dimension of \( \mathcal{F} \) rather than the growth function of \( \mathcal{F} \).

**Theorem 1.6** (Vapnik and Chervonenkis [53]) Let \( d \in \mathbb{N} \) given integer. There exists a constant \( \kappa \) such that with probability at least \( 1 - \delta \) over the random choice of \( S^m \) according to \( \mathbb{P}^m(x, y) \), every \( f \in \mathcal{F} \) with \( \text{VCdim}(\mathcal{F}) = d \) satisfies

\[
R_P[f] \leq R_{\text{emp}}[f, S^m] + \kappa \sqrt{\frac{d \ln(m/d) + \ln(1/\delta)}{m}},
\]

(1.7)

for \( m \geq d/2 \).

We note that the bound is independent of \( \mathbb{P}^m(x, y) \). It assumes only that both the training data and the test data are drawn independently according to some unknown \( P(x, y) \). Furthermore, this bound applies to all function classes \( \mathcal{F} \) with finite VC dimension, and is effectively calculable.

**Structural risk minimization**

In order to analyze the bound in (1.7) more efficiently, we name the whole upper bound the *guaranteed risk* and the second term on the right hand side the *confidence term*. One can see that the *expected risk* and the *empirical risk* depend on a specific
function \( f \) from the function class \( \mathcal{F} \), but the confidence term depends on the VC dimension of the whole function class. One would like to find that subset of \( \mathcal{F} \), such that the guaranteed risk bound for that subset is minimized. To do so, a "structure" is introduced by dividing the entire class of functions into nested subsets, see Figure 1.1. For each subset, we must be able either to compute \( VCDim(\mathcal{F}) \), or to get a bound on \( VCDim(\mathcal{F}) \) itself. The principle of structural risk minimization (SRM) [50, 51] then consists of finding that subset of functions which minimizes the guaranteed risk bound and the particular hypothesis for the subset. This can be done by simply training a series of machines, one for each subset, where for a given subset, the goal of training is simply to minimize the empirical risk. One then takes that trained machine in the series for which the right hand side in inequality (1.7) is minimal.

There is also a lower bound with a similar form as Theorem 1.5 that applies to a hypothesis selected by any inductive principle [2, 18]. These two bounds show that the generalization ability of a hypothesis chosen via ERM lies within a confidence
interval of the expected risk $R_p[f]$. Although these results are asymptotically close to the best possible, the practical performance exhibited by learning algorithms is generally much better than these theoretical results would suggest. One reason for this difference is that the VC dimension is sensitive to behavior on an arbitrarily small scale. For instance, the VC dimension of the hypothesis class $\mathcal{F}$ of the straight lines in $\mathbb{R}^2$ is 3. If a high frequency sine function with scale $\delta > 0$ is added on the function $f \in \mathcal{F}$ and constructs a new hypothesis class $\mathcal{F}^*$, one can now shatter any $m$ points well using $f^* \in \mathcal{F}^*$. So the VC dimension of $\mathcal{F}^*$ is infinity for all $\delta > 0$. In this example, although $\mathcal{F}^* \rightarrow \mathcal{F}$ as $\delta \rightarrow 0$, $VCdim(\mathcal{F}^*) \neq VCdim(\mathcal{F})$. In other words, two hypothesis classes with arbitrarily close decision boundaries have quite different VC dimensions. So the VC dimension is not a good measure of the generalization performance in this case.

In the following section, we will show that with the consideration of margin and the covering numbers of function classes (precise definitions are given below), this problem can be solved, and more robust bounds can be achieved in terms of the margin and covering numbers.

1.2.2 Covering numbers, margin and generalization performance

In the Definition 1.2 of the growth function, the function $f \in \mathcal{F}$ maps into $\{-1, 1\}$, so $\Pi_{\mathcal{F}}(m)$ is finite for every finite set $S$ for binary pattern recognition. But $\Pi_{\mathcal{F}}(m)$ may be infinite for a regression model where $\mathcal{F}$ is a class of real valued functions $f : \mathcal{X} \rightarrow \mathbb{R}$. In this case, one can use covering numbers to measure the effective "size" of $\mathcal{F}$ based on a given scale $\epsilon$. In order to define covering numbers, we will need more notation.

Spaces and norms

For $1 \leq p \leq \infty$ and $d \in \mathbb{N}$, define the spaces

$$
\ell_p^d := \{x \in \mathbb{R}^d : \|x\|_{\ell_p^d} < \infty\}
$$
CHAPTER 1. INTRODUCTION

UNIT BALL

Figure 1.2: Operator norm. Both $T_1$ and $T_2$ have the same norm.

where the $p$-norms are

$$\|x\|_{\ell^p_p} := \left( \sum_{j=1}^{d} |x_j|^p \right)^{1/p}, \text{ for } 1 \leq p < \infty;$$

$$\|x\|_{\ell^p_\infty} := \sup_{j=1,...,d} |x_j|, \text{ for } p = \infty.$$  

For $1 \leq p \leq \infty$, we write $\ell_p = \ell^\infty_p$ for $d = \infty$ and the norms are defined similarly.

The $\ell^d_\infty$ norm with respect to $X^m$ of $f \in \mathcal{F}$ is defined as

$$\|f\|_{\ell^\infty^m} := \max_{i=1,...,m} |f(x_i)|.$$  

Likewise, $\|f\|_{\ell^p^m} := \|(f(x_1),...,f(x_m))\|_{\ell^p_p}$. To simplify notation, we use $\ell^X^m_p$ for $1 \leq p \leq \infty$ to denote both the space and the metric induced by the norm in that space.

Assume $\mathcal{X}$ is a measurable space, given a measure $\mu$ on $\mathcal{X}$, some $1 \leq p < \infty$ and a function $f: \mathcal{X} \to \mathbb{K}$ we define

$$\|f\|_{L_p(\mathcal{X},\mathbb{K})} := \left( \int |f(x)|^p d\mu(x) \right)^{1/p}$$

if the integral exists and

$$\|f\|_{L_\infty(\mathcal{X},\mathbb{K})} := \text{ess sup}_{x \in \mathcal{X}} |f(x)|.$$  

For $1 \leq p \leq \infty$, we let

$$L_p(\mathcal{X},\mathbb{K}) := \{ f: \mathcal{X} \to \mathbb{K} : \|f\|_{L_p(\mathcal{X},\mathbb{K})} < \infty \}.$$
We let $L_p(\mathcal{X}) := L_p(\mathcal{X}, \mathbb{R})$.

Let $\mathcal{L}(E, F)$ be the set of all bounded linear operators $T$ between the normed spaces $(E, \| \cdot \|_E)$ and $(F, \| \cdot \|_F)$, i.e. operators such that the image of the (closed) unit ball

$$U_E := \{ x \in E : \|x\|_E \leq 1 \}$$

is bounded. The smallest such bound is called the operator norm (see Figure 1.2),

$$\|T\| := \sup_{x \in U_E} \|Tx\|_F.$$

In the following, $E$ and $F$ will always be Banach spaces, i.e. complete normed spaces (for instance $\ell_p^d$ spaces, $p \geq 1$). In some cases, they will be Hilbert spaces $H$, i.e. Banach spaces endowed with a dot product $\langle \cdot, \cdot \rangle_H$ giving rise to its norm via $\|x\|_H = \sqrt{\langle x, x \rangle_H}$.

Covering numbers

Let us define covering number now.

**Definition 1.7 (ϵ-covering number)** Given a set $S$, a metric $\rho$ on $S$, and $\epsilon > 0$. Suppose $\mathcal{F}$ is a subset of the metric space $(S, \rho)$. Then the $\epsilon$-covering number $\mathcal{N}(\epsilon, \mathcal{F}, \rho)$ is defined as the smallest number of balls of radius $\epsilon$ whose union contains $\mathcal{F}$ using the metric $\rho$.

One can define covering numbers in terms of different metrics $\rho$. We normally choose it as $\ell_\infty^m$. Figure 1.3 shows an $\epsilon$-cover of a set in $\mathbb{R}^2$.

This definition suffices if we have a fixed training set $S^m$. By dropping the dependence on $S^m$, and instead taking a maximum over all sets $S$ of size $m$ from the input space $\mathcal{X}$, we can obtain the uniform covering number of the hypothesis class $\mathcal{F}$.

**Definition 1.8 (Uniform covering number)** Given the same parameters as Definition 1.7, we define the uniform covering number of the function class $\mathcal{F}$ on $\mathcal{X}$ as

$$\mathcal{N}^m(\epsilon, \mathcal{F}) := \sup_{\mathcal{X}^m \in \mathcal{X}^m} \mathcal{N}(\epsilon, \mathcal{F}, \ell_\infty^m).$$

(1.9)
Let us calculate the uniform covering number of the function class \( \mathcal{F} \) of classifiers \( f : \mathcal{X} \to \{-1, 1\} \). Given a finite set \( \mathbf{X}^m \) of samples \( (x_1, \ldots, x_m) \), where \( x_i \in \mathcal{X} \) for \( i = 1, \ldots, m \), then \( \mathcal{F}|_{\mathbf{X}^m} \) is finite. Furthermore, for all \( 0 < \epsilon < 1 \),

\[
N(\epsilon, \mathcal{F}, \mathbf{X}^m) = |\mathcal{F}|_{\mathbf{X}^m}, \quad \forall \mathbf{X}^m \in \mathcal{X}^m.
\]

From Definitions 1.2 and 1.8, we will get \( N^m(\epsilon, \mathcal{F}) = \Pi_{\mathcal{F}}(m) \). This shows that the uniform covering number can be understood as a generalization of the growth function.

Next we will show that, as a measure of the richness of the class \( \mathcal{F} \) at the scale \( \epsilon \), the uniform covering number \( N^m(\epsilon, \mathcal{F}) \) determines the generalization performance attainable using \( \mathcal{F} \) (Theorem 1.11). But before that, we need to introduce another important notion, the *margin*.

**Margin**

As noted above, the outputs of the regression algorithm are real valued. In fact, many pattern recognition algorithms also produce their output by thresholding a real valued function, such as neural networks, support vector machines, and combined classifiers produced by voting methods. It is obvious that it is often useful to deal with the continuous function directly, since the thresholded output contains less information. So it is easy to imagine that the real-valued output can be interpreted as a measure of
generalization performance in pattern recognition. Such generalization performance is expressed in terms of margin. Next we define the margin of an example:

**Definition 1.9 (Margin)** Given an example \((x, y) \in \mathbb{R}^d \times \{-1, 1\}\) and a real-valued hypothesis \(f : \mathbb{R}^d \rightarrow \mathbb{R}\), the margin of \((x, y)\) with respect to \(f\) is defined as \(yf(x)\).

From this definition, we can see that an example is classified correctly if and only if it has a positive margin. A large value of the margin indicates that there is little uncertainty in the classification of the point. Thus, we would expect that a hypothesis with large margins would have good generalization performance. In fact, it can be shown that achieving a large margin on the training set results in an improved bound on the generalization performance (see, e.g., Theorem 1.12 below). Geometrically, for “well behaved” functions \(f\), the distance between a point and the decision boundary will roughly correspond to the magnitude of the margin at that point.

**Empirical margin risk**

With the consideration of margin, we can design another kind of empirical risk functional named *empirical margin risk*. In this risk functional, we include not only those samples that are classified wrongly, but also those samples that are nearly classified wrongly.

**Definition 1.10 (Empirical margin risk)** Given a training set \(S^m\) and a margin \(\gamma > 0\), for a function \(f\) in \(\mathcal{F}\), the empirical margin risk of \(f\) with respect to \(\gamma\) on \(S^m\) is

\[
R_{\text{emp}}^\gamma[f, S^m] = \frac{1}{m} |\{i : y_if(x_i) \leq \gamma\}|.
\]

So the empirical margin risk is clearly the proportion of examples with a margin less than or equal to \(\gamma\). It has the property that \(R_{\text{emp}}^\gamma[f, S^m]\) is nondecreasing with increasing margin \(\gamma\), and \(R_{\text{emp}}^0[f, S^m] = R_{\text{emp}}[f, S^m]\).
If a hypothesis estimates most training examples with a large margin, then we can approximate this hypothesis using a function from a simpler class. The margins of the training examples and the generalization performance are related [50, 2, 16]. We will determine the generalization performance of a hypothesis $f$ in terms of the margin empirical risk of $f$ and the covering numbers.

**Bounds using covering numbers**

There are many generalization error bounds expressed in terms of the empirical margin risk and $N^m(\epsilon, \mathcal{F})$. One result is derived in [2] by bounding the expectation of the value of the uniform covering numbers, and is converted in a straightforward manner to the form shown below.

**Theorem 1.11 (Convergence bound using covering numbers)**

Consider the hypothesis $f$ in a hypothesis space $\mathcal{F}$, a series of $m$ training examples $S^m \in (\mathcal{X} \times \mathcal{Y})^m$, and a margin $0 \leq \gamma < 1/2$. Then with probability at least $1 - \delta$ for any $0 < \delta < 1$,

$$R_P[f] \leq R_{\text{emp}}^*[f, S^m] + \sqrt{\frac{8}{m} \ln \left( \frac{2N^{2m}(\gamma/2, \mathcal{F})}{\delta} \right)}.$$  \hspace{2cm} (1.10)

Comparing Theorem 1.11 with the VC dimension bound of Theorem 1.6, we notice that this new result uses a scale sensitive dimension (the uniform covering numbers) instead of the VC dimension. Also, it uses the empirical margin risk, which depends on the proportion of the training set with margin less than some value $\gamma$. These are linked by the parameter $\gamma$, which appears both as the margin threshold for the empirical margin risk and (up to a constant) the scale for the uniform covering numbers.

**1.3 Maximal margin algorithms**

Theorem 1.11 bounds the generalization performance of a hypothesis $f$ in terms of the margin $\gamma$ with respect to the training set $S^m$. Such bounds motivate a kind of
1.3. MAXIMAL MARGIN ALGORITHMS

Figure 1.4: Maximal margin classification with margin $\gamma$. Hypothesis $f = 0$ is the classifier. $f = \gamma$ and $f = -\gamma$ are the edge of the band.

learning algorithm which separates the samples with the maximal margin hyperplane (Figure 1.4). Such an algorithm is called the maximal margin algorithm.

From Figure 1.4, we can see that slightly perturbing one training example with a large margin is unlikely to cause a change in the hypothesis $f$, and thus have little effect on the the generalization performance of $f$. There are two main maximal margin algorithms: support vector machines and boosting.

1.3.1 Support vector machines

Support vector machines [13, 16] are linear classifiers that use the maximal margin hyperplane in a feature space defined by a kernel function. Support vector machines have been successfully applied to many real world problems. We will give a brief introduction in this subsection. All facts mentioned here can be found in [16, 11].

Simple linearly separable case

Let us start from the simplest case of classification: linear decision functions trained on separable data (Figure 1.5). Suppose we have a hyperplane $f$ which separates the positive examples from the negative ones. The points which lie on the hyperplane
CHAPTER 1. INTRODUCTION

satisfy

\[ \langle w, x \rangle + b = 0, \]

where the weight vector \( w \) is perpendicular to the decision boundary induced by \( f \).

The perpendicular distance from \( f \) to the origin is \( |b|/\|w\| \), and \( \|w\| \) is the norm of \( w \). Following the above definition, the margin of an example \( x_i \) is \( y_i f(x_i) \). For the linearly separable case, the support vector algorithm simply looks for the separating hyperplane with maximum margin. Suppose the margin is \( \gamma = 1 \), all the training data should satisfy the following constraints:

\[
\begin{align*}
\langle w, x_i \rangle + b &\geq 1 & \text{for } y_i = +1 \\
\langle w, x_i \rangle + b &\leq -1 & \text{for } y_i = -1 
\end{align*}
\]

We can also find the relationship between margin \( \gamma \) and weight vector \( w \):

\[
\gamma = \frac{1}{2} \left( \frac{1-b}{\|w\|} - \frac{(-1-b)}{\|w\|} \right) = \frac{1}{\|w\|}.
\]

Thus we can find the hypothesis which gives the maximum margin by solving the following optimization problem

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|w\|^2, \\
\text{subject to} & \quad y_i (\langle w, x_i \rangle + b) \geq 1, \text{ for } i = 1, \ldots, m.
\end{align*}
\]

Figure 1.5 shows a typical two dimensional classification problem. Both the hyperplanes \( f_1 \) and \( f_2 \) can separate all the examples correctly, but only \( f_1 \) is the one with maximum margin. The training points for which the equality in (1.11) holds, and whose removal would change the hyperplane \( f_1 \), are called \textit{support vectors}. They are indicated in Figure 1.5 by the extra circles.

By constructing a Lagrange function from both the objective function and the corresponding constraints (for details, see e.g., [16]) and using the Karush-Kuhn-Tucker (KKT) conditions of optimization theory [7], the optimization problem (1.12) can be transferred to another optimization problem: find multipliers \( \alpha_i \) which

\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle, \\
\text{subject to} & \quad \alpha_i \geq 0, \\
& \quad \sum_{i=1}^{m} \alpha_i y_i = 0, \quad \text{for } i = 1, \ldots, m.
\end{align*}
\]
1.3. MAXIMAL MARGIN ALGORITHMS

Figure 1.5: A binary classification problem. The hyperplane $\mathbf{f}_1$ is the maximal margin classifier. When the black dot is ‘-’, it is a separable case. When the black dot is ‘+’, it is a non-separable case.

In (1.13), the constraints are rather more manageable than those in (1.12). And we can get the hyperplane decision function as

$$ f(x) = \text{sign} \left( \sum_{i=1}^{m} y_i \alpha_i (x, x_i) + b \right). \quad (1.14) $$

The value of $b$ does not appear in the optimization problem (1.13). We can use the primal constraints to find it:

$$ b = -\frac{\max_{y_j = -1} (\sum_{i=1}^{m} y_i \alpha_i (x_j, x_i)) + \min_{y_j = 1} (\sum_{i=1}^{m} y_i \alpha_i (x_j, x_i))}{2}. \quad (1.15) $$

Non-separable cases

In practice, there are many non-separable cases. For example, if the value of the black dot in Figure 1.5 changes from negative to positive, the problem will become non-separable. In order to use the maximal margin algorithm to solve problems of this type, we need to relax the constraints in (1.11). This can be done by introducing
positive slack variables $\xi_i$, $i = 1, \ldots, m$, in the constraints (see [13] for detail), which become
\[
\begin{align*}
\langle w, x_i \rangle + b & \geq +1 - \xi_i \quad \text{for } y_i = +1 \\
\langle w, x_i \rangle + b & \leq -1 + \xi_i \quad \text{for } y_i = -1 \\
\xi_i & \geq 0 \quad \forall i.
\end{align*}
\tag{1.16}
\]

Thus, for an error to occur, the corresponding $\xi_i$ must exceed unity, so $\sum_i \xi_i$ is an upper bound on the number of training errors. Hence a natural way to assign an extra cost for errors is to change the optimization problem to
\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2}\|w\|^2 + C \sum_{i=1}^{m} \xi_i, \\
\text{subject to} & \quad y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i, \quad \text{for } i = 1, \ldots, m,
\end{align*}
\tag{1.17}
\]
where $C$ is a parameter to be chosen by the user, a larger $C$ corresponding to assigning a higher penalty to errors. For a particular problem, choosing a particular value for $C$ corresponds to choosing a value for $\|w\|$, and then minimizing $\|\xi\|$ for that size of $w$.

Again, if we construct a Lagrange function from both the objective function and the corresponding constraints and use the KKT conditions, the optimization problem (1.17) can be transformed to the following optimization problem: find multipliers $\alpha_i$ which
\[
\begin{align*}
\text{maximize} & \quad \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle, \\
\text{subject to} & \quad C \geq \alpha_i \geq 0, \\
& \quad \sum_{i=1}^{m} \alpha_i y_i = 0, \quad \text{for } i = 1, \ldots, m.
\end{align*}
\tag{1.18}
\]
Thus the only difference from the separable case is that the $\alpha_i$ now have an upper bound of $C$.

Feature space and kernels

If we ignore the details of the calculation of $\alpha_i$ in the optimization problem (1.13) or (1.18), we notice that there is one crucial property of the algorithm that we need to emphasize: both the optimization problem and the final decision function (1.14) depend only on the inner products between inputs: $\langle x_i, x_j \rangle$. Suppose we map the
data to some other inner product space $\mathcal{S}$ via a nonlinear map which we will call $\Phi$:

$$
\Phi: \mathbb{R}^d \to \mathcal{S}.
$$

(1.19)

Now the above linear algorithm will operate in $\mathcal{S}$, which, as we have noted, would only depend on the data through inner products in $\mathcal{S}$: $\langle \Phi(x_i), \Phi(x_j) \rangle$. Clearly, if $\mathcal{S}$ is high-dimensional, this inner product will be very expensive to compute. Boser, Guyon and Vapnik [8] show that a rather old trick [1] can be used to accomplish this in an astonishingly straightforward way. That is, there is a simple function $k$ such that

$$
k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle,
$$

(1.20)

which can be evaluated efficiently. So we only need to use $k$ in the optimization algorithm and never need to explicitly know what $\Phi$ is. We call $k$ a kernel and $\Phi$ the feature space. We also define the corresponding kernel matrix $K$ by $K_{ij} := k(x_i, x_j)$.

The first kernels investigated for support vector machines are the polynomial kernels

$$
k(x_i, x_j) = (\langle x_i, x_j \rangle + b)^p,
$$

(1.21)

the radial basis functions (RBF)

$$
k(x_i, x_j) = \exp(-\|x_i - x_j\|^2/(2\delta^2)),
$$

(1.22)

and sigmoid kernels

$$
k(x_i, x_j) = \tanh(\kappa(x_i, x_j) + b).
$$

(1.23)

**Support vector machines**

Now we have the tools to construct a nonlinear classifier. To do this, we substitute $\Phi(x_i)$ for each training example $x_i$, and perform the optimal hyperplane algorithm in $\mathcal{S}$. Because we are using kernels, we can thus get the nonlinear decision function $f$ as

$$
f(x) = \text{sign} \left( \sum_{i=1}^{m} y_i \alpha_i k(x, x_i) + b \right).
$$

(1.24)
Compared with equation (1.14), the only difference is the inner product in (1.14) now becomes the kernel \( k(x, x_i) \). The parameters \( \alpha_i \) are still computed as the solution of a quadratic programming problem.

For the regression case, the algorithm tries to construct a linear function in the feature space such that the training points lie within a distance \( \xi > 0 \). Similar to the pattern recognition case, this can also be written as a quadratic programming problem in terms of kernels. The nonlinear regression estimate of support vector machines takes the form

\[
f(x) = \sum_{i=1}^{m} (\alpha_i - \tilde{\alpha}_i) k(x, x_i) + b,
\]

where \( \alpha_i \) and \( \tilde{\alpha}_i \) are the dual variables, which are also computed as the solution of the corresponding quadratic programming problem (see Section 6.2 in [16]).

### 1.3.2 Boosting

Boosting is a general method for improving the accuracy of a learning algorithm. Schapire [36] was the first to pose the question of whether a weak learning algorithm which performs just slightly better than random guessing can be voted into an arbitrarily accurate strong learning algorithm. In recent years, many researchers have reported significant improvements in the generalization performance using boosting methods with learning algorithms such as C4.5 [32] or CART [10] as well as with neural networks [5].

A number of popular and successful boosting methods can be seen as gradient descent algorithms, which implicitly minimize some cost function of the margin [9, 20, 30]. In particular, the popular AdaBoost algorithm [19] can be viewed as a procedure for producing voted classifiers which minimize the sample average of an exponential cost function of the training margins.
A brief overview of boosting

Let us give a more precise description of boosting in the binary case now. The aim of boosting algorithms is to give a hypothesis which is a voted combination of classifiers of the form \( \text{sign}(f(x)) \), with

\[
f(x) = \sum_{t=1}^{T} \alpha_t h_t(x),
\]

where \( \alpha_t \in \mathbb{R} \) are the classifier weights, \( h_t \) are base classifiers from some class \( \mathcal{F} \) and \( T \) is the number of base classifiers chosen from \( \mathcal{F} \). Boosting algorithms take the approach of finding voted classifiers which minimize the sample average of some cost function of the margin. That is, for a training set \( S^m \), one wants to find \( f \) such that

\[
R_{\text{emp}}[f, S^m] := \frac{1}{m} \sum_{i=1}^{m} c(y_i, f(x_i))
\]

is minimized for some suitable cost function \( c: \{-1, 1\} \times \mathbb{R} \rightarrow \mathbb{R} \). For AdaBoost [19], the corresponding cost function is:

\[
c(y_i, f(x_i)) = e^{-y_i f(x_i)},
\]

The base hypothesis \( h_t: \mathcal{X} \rightarrow \{-1, 1\} \) and the combinations \( f \) are viewed as elements of an inner product space. For a boosting algorithm, this inner product can be defined by

\[
\langle f, g \rangle := \frac{1}{m} \sum_{i=1}^{m} f(x_i)g(x_i)
\]

for all real functions \( f, g \) defined on \( \mathcal{X} \).

Nearly all the boosting algorithms iteratively construct the combination one classifier at a time. So we will denote the combination of the first \( t \) classifiers by \( f_t \), while the final combination of \( T \) classifiers will simply be denoted by \( f \).

Generalization performance of boosting

In [37], Schapire et al. show that boosting is good at finding classifiers with large margins in that it concentrates on those examples whose margins are small (or negative) and forces the base learning algorithm to generate good classifications for those
examples. Thus, boosting can effectively find a large margin hyperplane. Furthermore, the generalization error bound of the large margin hypothesis, which is a voted combination of classifiers, was given in [37].

**Theorem 1.12** Let \( P(x, y) \) be a distribution over \( \mathcal{X} \times \{-1, 1\} \), and let \( S^m \) be a sample of \( m \) examples chosen independently at random according to \( P \). Suppose the base-classifier space \( \mathcal{F} \) has VC-dimension \( d \), and let \( \delta > 0 \). Assume that \( m \geq d \geq 1 \). Then with probability at least \( 1 - \delta \), every weighted average function \( f \), satisfies the following bound for all margin \( \gamma > 0 \):

\[
R_P[f] \leq R_{\text{emp}}[f, S^m] + O \left( \sqrt{\frac{d \log^2(m/d)}{m \gamma^2}} + \frac{\log(1/\delta)}{m} \right). \tag{1.29}
\]

This result, like Theorems 1.6 and 1.11, bounds the generalization error using two terms. The first term depends on the empirical margin risk. The second term depends on the VC dimension of the class of base classifiers, but this time with a penalty on the size of \( \gamma \). Also note that the second term does not depend on \( T \), the number of classifiers in the combination. The bound holds for any choice of the margin \( \gamma \).

### 1.4 Outline of the thesis

In Chapter 2, we present a new formula for bounding the covering numbers of support vector machines following the result in [60]. The particular advantage of this new bound is that it is directly in terms of the eigenvalues of the integral operator induced by the kernel. Following the theorem of the main result, we illustrate the new result by bounding the covering numbers of support vector machines which use gaussian radial basis function (RBF) kernels. We show that the new bound is easily computed and considerably better than previous results that did not take account of the kernel. The result shows the influence of the kernel width on the covering numbers: the covering number bound decreases when the width increases. More generally, the main result makes model order selection possible using any parameterized family of
kernel functions, since it describes how the capacity of the class is affected by changes to the kernel.

Furthermore, there is an important number $j^*$ in the main result. We explain $j^*$ as the effective dimension of the function class, which can illustrate the character of kernel expansions clearly. For a smooth kernel, the “effective dimension” $j^*$ is small. The value of $j^*$ depends on the covering number of the function class. Thus $j^*$ can be considered analogous to existing “scale-sensitive” dimensions, such as the fat-shattering dimension. Note the bounds for $j^*$ explicitly depend on the kernel.

In Chapter 3, we summarize the classical support vector classification and regression algorithms systematically. We then apply support vector algorithms for two nonlinear time series prediction problems. The experiments examine the effect of different cost functions and different kernels. For the mobile radio signal prediction, which is normally solved by linear methods, support vector algorithms also show good performance when nonlinear kernels are used.

In Chapter 4, we describe a class of algorithms (which we call NormBoost algorithms) which minimize a regularized risk functional that includes an error term and a regularization term. Here the error term may be, for instance, the original AdaBoost cost function, and the regularization term is a function either defined in the Hilbert space of $\mathcal{H}$ or a RKHS. Like AdaBoost, NormBoost performs gradient descent so as to maximally reduce the regularized risk functional at each iteration. At the same time, due to the regularization term, NormBoost approximates a smoother function than AdaBoost.

We first introduce the idea of gradient descent, and the class of NormBoost algorithms for different regularization terms. Each algorithm chooses linear combinations of elements of an inner product space so as to minimize different regularized risk functions. Then we present experimental results for the NormBoost algorithms. These experiments show that the new algorithms based on RKHS norms progressively outperform AdaBoost when the label noise increases.

For the RKHS based regularization, we can appeal to the “representer theorem”
which allows us to conclude that the optimal linear combination of weak hypotheses can actually be written using only $m$ terms (where $m$ is the number of examples). The significance of this is that it allows us to perform gradient descent in an $m$-dimensional space rather than effectively in an infinite dimensional one, as for the standard boosting algorithms. One would expect that this leads to faster convergence, and indeed we demonstrate a marked improvement in convergence rate.

Finally, we use the theoretical results due to [37] to bound the error of a combination of classifiers in terms of the norm bound of the combined classifier.

In Chapter 5, we present conclusions and outline the future work.
Chapter 2

Covering Numbers for Support Vector Machines

Until recently, the only bounds on the generalization performance of support vector machines (within Valiant's probably approximately correct framework) took no account of the kernel used except through its effect on the margin and radius. It has been shown [60] that one can bound the covering numbers relevant for such bounds using tools from functional analysis. In this chapter, we show that the resulting bound in [60] can be greatly simplified. The new bound involves the eigenvalues of the integral operator induced by the kernel. It shows that the "effective dimension" of the class of functions induced by the kernel depends on the rate of decay of these eigenvalues. We present an explicit calculation of covering numbers for a support vector machine using a gaussian kernel, which is significantly better than that implied by previous results.

2.1 Introduction and previous results

As we described in Chapter 1, in the traditional viewpoint of statistical learning theory, the generalization performance attainable using a class of functions $\mathcal{F}$ is determined via the covering numbers $\mathcal{N}(\epsilon, \mathcal{F})$ (see e.g., Theorem 1.11). Support vector
machines are learning algorithms based on maximal margin hyperplanes. Consequently one can apply an analysis for the maximal margin algorithm directly to support vector machines. However such a process ignores the effect of the kernel. Intuitively one would expect that a "smoother" kernel would somehow reduce the capacity of the learning machine thus leading to better bounds on generalization error if the machine can attain a small training error.

In [60, 58] it has been shown that this intuition is justified. The main result there gives a bound on the covering numbers for the class of functions computed with support vector machines such as Theorem 1.11. This bound along with statistical results in [4, 41] results in bounds that explicitly depend on the kernel used. The intuitive idea that eigenvalues of kernels must have something to say about generalization performance has also been previously explored by others in a different analysis framework: see the (simultaneous and independent) development in terms of regularization theory in [44] and [21]. One can also recover a dependence of covering numbers on eigenvalues in a different setting: in [42] it was shown how the eigenvalues of the empirical gram matrix can bound the empirical covering numbers and in turn how generalization results can be obtained that way.

In [60, 58], the function class \( \mathcal{F} \) of support vector machines is viewed as being generated by an integral operator induced by the kernel, and properties of this operator are used to bound the required covering numbers. We will briefly introduce this new bound in this section.

**Kernels, eigenvectors and eigenvalues**

We already know that support vector machines are learning algorithms which make use of an implicit nonlinear map \( \Phi \) from the input space \( \mathcal{X} \) into the feature space \( \mathcal{S} \) by using a general kernel function

\[
k(x_i, x_j) := \langle \Phi(x_i), \Phi(x_j) \rangle
\]

in place of the standard inner product \( \langle x_i, x_j \rangle \). Examples are given in (1.21), (1.22) and (1.23). The benefit of such kernel functions is that we do not need to know the
exactly form of map \( \Phi \), and need only use \( k \) in the algorithm.

So the following question arises, what kinds of kernel are suitable for support vector machines and which are not? Mercer's condition provides an answer [51, 14]. Before the introduction of Mercer's Theorem, we need to define more notations.

Suppose \( T : E \rightarrow E \) is a linear operator mapping a normed space \( E \) into itself. We say that \( x \in E \) is an eigenvector of \( T \) if for some scalar \( \lambda \), \( Tx = \lambda x \) where \( x \neq 0 \). Such a \( \lambda \) is called the eigenvalue associated with \( x \). When \( E \) is a function space (e.g., \( E = L_2(\mathcal{X}) \)) the eigenvectors are of course functions, and are usually called eigenfunctions. Thus \( \psi_n \) is an eigenfunction of \( T : L_2(\mathcal{X}) \rightarrow L_2(\mathcal{X}) \) if \( T\psi_n = \lambda \psi_n \).

In general \( \lambda \) is complex, but in this chapter all eigenvalues are real (because of the symmetry of the kernels used to induce the operators). The inner product in \( L_2(\mathcal{X}) \) is defined as \( \langle f, g \rangle = \int_{\mathcal{X}} f(\tau)g(\tau) d\tau \).

Let \( \lambda_1 \geq \lambda_2 \geq \cdots \geq 0 \) be the eigenvalues of the integral operator

\[
T_k : L_2(\mathcal{X}) \rightarrow L_2(\mathcal{X})
\]

\[
T_k : f \mapsto \int_{\mathcal{X}} k(\cdot, y)f(y)dy,
\]

and denote by \( \psi_n(\cdot) \), \( n \in \mathbb{N} \) the corresponding eigenfunctions. (The eigenvalues are real and nonnegative because of the assumptions on \( k \) — see Mercer's Theorem below.) For translation invariant kernels (such as \( k(x, y) = \exp((x - y)^2/\sigma^2) \)), the eigenvalues are given by

\[
\lambda_i = \sqrt{2\pi} K(j\omega_0) \tag{2.2}
\]

for \( j \in \mathbb{Z} \), where \( K(\omega) = F[k(x)](\omega) \) is the Fourier transform of \( k(\cdot) \) (see [60, 58] for further details; and see Section 2.3 for an explanation of \( \omega_0 \)). For smoother kernels, the Fourier transform \( F(j\omega_0) \) decreases faster. (There are less "high frequency components.""

Thus for smooth kernels, \( \lambda_i \) decreases to zero rapidly for increasing \( i \).

Mercer's condition

Now let us introduce Mercer's Theorem. The version stated below is a special case of the theorem proven in [28, p. 145].
Theorem 2.1 (Mercer’s Theorem) Suppose $k \in L_{\infty}(\mathcal{X} \times \mathcal{X})$ is a symmetric kernel (i.e., $k(x, x') = k(x', x)$) such that the integral operator $T_k : L_2(\mathcal{X}) \to L_2(\mathcal{X})$,

$$T_k f(\cdot) := \int_{\mathcal{X}} k(\cdot, y) f(y) dy$$

(2.3)

is positive (i.e., for all $f \in L_2(\mathcal{X})$, $(T_k(f), f) \geq 0$; for symmetric $k$ this is equivalent to $\lambda_i \geq 0$ for all $i$). Let $\psi_j \in L_2(\mathcal{X})$ be the eigenfunction of $T_k$ associated with the eigenvalue $\lambda_j \neq 0$ and normalized such that $\|\psi_j\|_{L_2} = 1$ and let $\overline{\psi_j}$ denote its complex conjugate. Suppose $\psi_j$ is continuous for all $j \in \mathbb{N}$. Then

1. $(\lambda_j(T))_j \in \ell_1$.
2. $k(x, y) = \sum_{j \in \mathbb{N}} \lambda_j \psi_j(x) \psi_j(y)$ holds for all $(x, y)$, where the series converges absolutely and uniformly for almost all $(x, y)$.

We will call a kernel satisfying the conditions of this theorem a Mercer kernel.

Here the term almost all means that the set of measure zero is not included. For example, while using a Lebesgue-Borel measure, the measure of a countable set of individual points is zero.

Note that there is an incorrect additional conclusion of Mercer’s theorem in [28, p. 145]

$$C_k := \sup_j \|\psi_j\|_{L_\infty} < \infty.$$  (2.4)

Schölkopf et al. give an example of kernel for which the condition (2.4) is not satisfied in [40, p. 58]. In [40, p. 58], it is also shown that the next condition is true

$$l_j := \sup_{x \in \mathcal{X}} \lambda_j |\psi_j(x)|^2 < \infty.$$  (2.5)

If we assume $C_k$ exists, i.e. inequality (2.4) holds, we have $l_j \leq \lambda_j C_k^2$, and $l_j$ decay as fast as $\lambda_j$. For simplicity of presentation, we assume $C_k < \infty$. We are unaware of any kernel used in practice for which $C_k$ is infinite. Strictly speaking, we should consider the $l_j$ instead of the $\lambda_j$ in the following material.
Support vector machines function class

Since we are interested in the generalization performance of support vector machines, we give an explicit definition of the corresponding hypothesis class.

Let \( k: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \) be a kernel satisfying the hypotheses of Mercer's theorem. Given \( m \) points \( x_1, \ldots, x_m \in \mathcal{X} \) as the input data, we will map the input data into a feature space \( \mathcal{S} \) (which is in fact a Hilbert space) via a mapping \( \Phi \). We let \( \tilde{x} := \Phi(x) \), and denote by \( \mathcal{F}_{R_w}(X^m) \) the hypothesis class implemented by support vector machines on an \( m \)-sample with weight vector (in feature space \( \mathcal{S} \)) bounded by \( R_w \):

\[
\mathcal{F}_{R_w}(X^m) := \{ X^m \mapsto (\langle w, \tilde{x}_1 \rangle, \ldots, \langle w, \tilde{x}_m \rangle) : x_i \in X^m, X^m \in \mathcal{X}^m, w \in \mathcal{S}, \|w\| \leq R_w \},
\]

and the hypothesis class \( \mathcal{F}_{R_w} \) on \( \mathcal{X}^m \) is defined as

\[
\mathcal{F}_{R_w} = \bigcup_{m=1}^{\infty} \bigcup_{X^m \in \mathcal{X}^m} \mathcal{F}_{R_w}(X^m).
\]

Here \( \langle \cdot, \cdot \rangle \) is the inner product in \( \mathcal{S} \).

Entropy number

Instead of giving the bound of covering numbers directly, another measurement of the effective capacity of a class of functions \( \mathcal{F} \), the entropy number is used in the analysis. So let us define it now.

**Definition 2.2 (Entropy number)** Given the normed space \( (E, \| \cdot \|_E) \) and a set \( M \subset E \). The \( n \)th entropy number of a set of \( M \), for \( n \in \mathbb{N} \), is

\[
e_n(M) := \inf \{ \epsilon > 0 : \text{there exists an } \epsilon\text{-cover for } M \text{ in } E \text{ containing } n \text{ or fewer points} \}.
\]

Let \( \mathcal{L}(E, F) \) be the set of all bounded linear operators \( T \) between the normed spaces \( (E, \| \cdot \|_E) \) and \( (F, \| \cdot \|_F) \). The entropy numbers of an
operator \( T \in \mathcal{L}(E, F) \) are defined as

\[
\epsilon_n(T) := \epsilon_n(T(U_E)). \tag{2.9}
\]

Note that \( \epsilon_1(T) = \|T\| \), and that \( \epsilon_n(T) \) certainly is well defined for all \( n \in \mathbb{N} \) if \( T \) is a compact operator, i.e. if \( T(U_E) \) is compact. In Figure 1.2, although \( T_1 \) and \( T_2 \) have the same norm, they are quite different operators. From the definition, the entropy numbers can be used to detect the difference: \( \epsilon_n(T_1) > \epsilon_n(T_2) \) when \( n > 1 \). Thus the entropy number \( \epsilon_n(T) \) gives more information about \( T \).

The dyadic entropy numbers of an operator are defined by

\[
\epsilon_n(T) := \epsilon_{2^n-1}(T), \quad n \in \mathbb{N}; \tag{2.10}
\]

similarly, the dyadic entropy numbers of a set are defined from its entropy numbers. A very nice introduction to entropy numbers of operators is [12].

The function \( n \mapsto \epsilon_n(M) \) can be thought of as the functional inverse of the function \( \epsilon \mapsto \mathcal{N}(\epsilon, M, d) \),

\[
\epsilon_n(\mathcal{F}_{Rw}) \leq \epsilon_0 \iff \mathcal{N}^m(\epsilon_0, \mathcal{F}_{Rw}) \leq n. \tag{2.11}
\]

**Bounds for support vector machines**

Next we give the bound of entropy numbers for support vector machines. See [60] for more detail.

From statement 2 of Theorem 2.1, it follows that \( k(x, y) \) corresponds to an inner product in \( \ell_2 \) i.e. \( k(x, y) = \langle \Phi(x), \Phi(y) \rangle_{\ell_2} \) with

\[
\Phi : \mathcal{X} \to \ell_2 \\
\Phi : x \mapsto (\phi_j(x))_j := (\sqrt{\lambda_j} \psi_j(x))_j \tag{2.12}
\]

for almost all \( x \in \mathcal{X} \). In the following we will (without loss of generality) assume the sequence of \( (\lambda_j)_j \) is sorted in non-increasing order. Assume (2.4) is satisfied. From the argument above one can see that \( \Phi(\mathcal{X}) \) lives not only in \( \ell_2 \) but in an axis parallel parallelepiped with lengths \( 2C_k \sqrt{\lambda_j} \). So let us consider maps that map \( \Phi(\mathcal{X}) \) into balls of some radius \( R \) centered at the origin.
2.1. INTRODUCTION AND PREVIOUS RESULTS

Figure 2.1: Mapping $\Phi$ into a ball of radius $R$.

Figure 2.2: Construct operator $A$ mapping from the unit ball to the ellipsoid which covers $\Phi$. So the entropy number of $A$ is the bound of the entropy number of $\Phi$.

Lemma 2.3 (Mapping $\Phi(x)$ into $\ell_2$) Let $S$ be the diagonal map

$$S : \mathbb{R}^N \to \mathbb{R}^N$$

$$S : (x_j)_j \mapsto S(x_j)_j = (s_j x_j)_j.$$  \hspace{1cm} (2.13)

Then $S$ maps $\Phi(\mathcal{X})$ into a ball of finite radius $R_S$ centered at the origin if and only if $(\sqrt{\lambda_j s_j})_j \in \ell_2$.

We can then construct a mapping $A$ from the unit ball in $\ell_2$ to an ellipsoid $\mathcal{E}$ such that $\Phi(\mathcal{X}) \subset \mathcal{E}$. From the definition of the entropy number of an operator (2.9), we know that $A$ will be useful for computing the entropy number of $\Phi(\mathcal{X})$. Figures 2.1 and 2.2 illustrate this explanation.

The operator $A$ that satisfies the above conditions is designed as

$$A : \ell_2 \to \ell_2$$

$$A : (x_j)_j \mapsto (R_A a_j x_j)_j.$$  \hspace{1cm} (2.14)
where $R_A := C_k \| (\sqrt{X_j/a_j})_j \|_{\ell_2}$.

By using the theorem due to Gordon, König and Schütt [23, p. 226] (stated in the present form in [12, p. 17]), we can exploit the freedom in choosing $A$ to minimize an entropy number as the following theorem shows. This will be a key ingredient of our calculation of the covering numbers for support vector machines.

**Theorem 2.4 (Entropy numbers for $\Phi(\mathcal{X})$)** Let $k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a Mercer kernel satisfying (2.4). Choose $a_j > 0$ for $j \in \mathbb{N}$ such that $(\sqrt{X_s/a_s})_s \in \ell_2$, and define $A: \ell_2 \to \ell_2$ by

$$A(x_j)_j = (R_A a_j x_j)_j$$

with $R_A := C_k \| (\sqrt{X_j/a_j})_j \|_{\ell_2}$, where $(x_j)_j$ is a sequence in $\ell_2$ and $x_j$ a real number respectively. Then

$$\epsilon_n(A) \leq \sup_{j \in \mathbb{N}} 6 C_k \left\| \left( \sqrt{X_s/a_s} \right)_s \left( \frac{a_1 \cdots a_j}{n} \right)^{1/j} \right\|_{\ell_2}.$$  

We will determine the uniform $\varepsilon$-covering numbers of $\mathcal{F}_{R_w}$ (see Definition 1.8), $N^m(\mathcal{F}_{R_w})$. In order to do so, let us design an operator $T$ to be

$$T = S_{\check{X}_m} R_w, \text{ where } R_w \in \mathbb{R},$$

where the operator $S_{\check{X}_m}$ is defined by

$$S_{\check{X}_m}: \ell_2 \to \ell^m_{1,\infty}$$

$$S_{\check{X}_m}: w \mapsto ((\check{x}_1, w), \ldots, (\check{x}_m, w)),$$

with $\check{x}_j \in \Phi(\mathcal{X})$ for all $j$. The following two lemmas are useful when computing entropy numbers in terms of $T$ and $A$.

**Lemma 2.5 (Carl and Stephani [12, p. 246])** Let $S \in \mathcal{L}(H, \ell^m_{1,\infty})$ where $H$ is a Hilbert space. Then there exists a constant $c > 0$ such that for all $m, n \in \mathbb{N}$

$$e_n(S) \leq c \|S\| \left( n^{-1} \log \left( 1 + \frac{m}{n} \right) \right)^{1/2}.$$
An alternative proof of this result (given in [59]) provides a small explicit value for the constant: \( c \leq 103 \). However there is reason to believe this bound is quite loose (see [59] for more detail).

According to \( \epsilon_{2n-1} = \epsilon_n \), we have
\[
\epsilon_n(S) \leq c\|S\| \left( (\log n + 1)^{-1} \log \left( 1 + \frac{m}{\log n + 1} \right) \right).
\]

Lemma 2.6 (Carl and Stephani [12, p. 11]) Let \( E, F, G \) be Banach spaces, \( R \in \mathcal{L}(F, G) \), and \( S \in \mathcal{L}(E, F) \). Then, for \( n, t \in \mathbb{N} \),
\[
\begin{align*}
\epsilon_{nt}(RS) &\leq \epsilon_n(R)\epsilon_t(S) \quad (2.19) \\
\epsilon_n(RS) &\leq \epsilon_n(R)\|S\| \quad (2.20) \\
\epsilon_n(RS) &\leq \epsilon_n(S)\|R\|. \quad (2.21)
\end{align*}
\]

Note that the latter two inequalities follow directly from the fact that \( \epsilon_1(R) = \|R\| \) for all \( R \in \mathcal{L}(F, G) \).

Now we can combine the bounds on entropy numbers of \( A \) and \( S^m_\mathcal{X} \) to obtain bounds for support vector machines.

Theorem 2.7 (Bounds for support vector classes) Let \( k \) be a Mercer kernel satisfying (2.4), let \( \Phi \) be induced via (2.12) and let the operators \( T \) and \( S^m_\mathcal{X} \) as defined in (2.17). Let \( A \) is defined as in Theorem 2.4 and suppose \( \tilde{x}_j = \Phi(x_j) \) for \( j = 1, \ldots, m \). Then the entropy numbers of \( T \) satisfy the following inequality:
\[
\epsilon_n(T : \ell_2 \rightarrow \ell^m_{\infty}) \leq R_w \inf_{(a_s)_s : (\sqrt{A_s/a_s})_s \in \ell_2} \epsilon_n(A), \quad (2.22)
\]

where \( A \) is defined as in Theorem 2.4. Notice that while \( \epsilon_n(T : \ell_2 \rightarrow \ell^m_{\infty}) \) depends on \( m \), its upper bound does not.

Combining Equations (2.16) and (2.22) gives effective bounds on \( N^m(\epsilon, \mathcal{F}_{R_w}) \) since (2.11) holds.

These results thus give a method to obtain bounds on the entropy numbers for kernel machines. In inequalities (2.16) and (2.22), we can choose \((a_s)_s\) and \( j \) to
optimize the bound. The key technical contribution of this chapter is the explicit determination of the best choice of \((a_s)_s\) and \(j\).

We assume \(a_s > 0\) for all \(s\). For \(j \in \mathbb{N}\), we define the set

\[
A_j = \left\{ (a_s)_s : \sup_{i \in \mathbb{N}} \left( \frac{a_1 \cdots a_i}{n} \right)^\frac{1}{i} = \left( \frac{a_1 \cdots a_j}{n} \right)^\frac{1}{j} \right\}. \tag{2.23}
\]

In other words, \(A_j\) is the set of \((a_s)_s\) such that the

\[
\sup_{i \in \mathbb{N}} \left( \frac{a_1 a_2 \cdots a_i}{n} \right)^\frac{1}{i}
\]

is attained at \(i = j\).

Let

\[
B((a_s)_s, n, j) = \left\| \left( \sqrt{\lambda_s/a_s} \right)_s \right\|_{\ell^2} \left( \frac{a_1 \cdots a_j}{n} \right)^\frac{1}{j}. \tag{2.24}
\]

### 2.2 Covering number bound

Our aim in this section is to show that the infimum in (2.22) and the supremum in (2.16) can be achieved and to give explicit expressions for the sequence \((a_s)_s\) and number \(j^*\) that achieve them. Following that, we can get a covering number bound which is easy to directly calculate.

#### 2.2.1 The optimal choice of \((a_s)_s\) and \(j\)

The main technical theorem is as follows.

**Theorem 2.8** Let \(k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}\) be a Mercer kernel. Suppose \(\lambda_1, \lambda_2, \ldots\) are the eigenvalues of \(T_k\). For any \(n \in \mathbb{N}\), the minimum

\[
j^* = \min \left\{ j : \lambda_{j+1} < \left( \frac{\lambda_1 \cdots \lambda_j}{n^2} \right)^\frac{1}{j} \right\} \tag{2.25}
\]

always exists, and

\[
\inf_{(a_s)_s : (\sqrt{\lambda_s/a_s})_s \in \ell^2} \sup_{j \in \mathbb{N}} B((a_s)_s, n, j) \leq B((a^*_s)_s, n, j^*),
\]

where \((a^*_s)_s\) is the sequence that achieves the infimum in (2.22) and \(j^*\) is the number that achieves the supremum in (2.16).
where
\[
a_i^* = \begin{cases} 
\sqrt{\lambda_i} & \text{when } i \leq j^* \\
\left(\frac{\sqrt{\lambda_1 \cdots \lambda_{j^*}}}{n}\right)^{\frac{1}{2}} & \text{when } i > j^*.
\end{cases}
\] (2.26)

This choice of \((a_s)_s\) results in a simple form for the bound of (2.22) in terms of \(n\) and \((\lambda_i)_i^\star\):

**Corollary 2.9** Let \(k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}\) be a Mercer kernel and let \(A: \ell_2 \to \ell_2\) be given by (2.15). Then for any \(n \in \mathbb{N}\), the entropy numbers satisfy
\[
\inf_{(a_s)_s, (\sqrt{\lambda_s}/a_s)_s \in \ell_2} \epsilon_n(A) \leq 6 C_k \sqrt{j^* \left(\frac{\lambda_1 \cdots \lambda_{j^*}}{n^2}\right)^{\frac{1}{2}}} + \sum_{i=j^*+1}^{\infty} \lambda_i,
\] (2.27)
with \(j^* = \min\left\{j : \lambda_{j+1} < \left(\frac{\lambda_1 \cdots \lambda_j}{n^2}\right)^{\frac{1}{2}}\right\} \).

### 2.2.2 Main result

Corollary 2.9, together with (2.22), implies the following theorem, which gives a covering number bound amenable to direct calculation.

**Theorem 2.10 (Main Result)** Suppose \(k\) is a kernel satisfying the hypothesis of Mercer’s Theorem. Let the hypothesis class \(\mathcal{F}_{R_w}\), eigenfunctions \(\psi_n(\cdot)\) and eigenvalues \((\lambda_i)_i^\star\) be defined as above. Suppose
\[
C_k := \sup_n \|\psi_n\|_{L_\infty} < \infty.
\] (2.28)

Then for \(n \in \mathbb{N}\) the minimum
\[
j_n^* = \min\left\{j : \lambda_{j+1} < \left(\frac{\lambda_1 \cdots \lambda_j}{n^2}\right)^{\frac{1}{2}}\right\}
\]
always exists. Define
\[
\epsilon_n^* = 6 R_w C_k \sqrt{j_n^* \left(\frac{\lambda_1 \cdots \lambda_{j_n^*}}{n^2}\right)^{\frac{1}{2}}} + \sum_{i=j_n^*+1}^{\infty} \lambda_i,
\] (2.29)
Then
\[ N^m(\epsilon^*_n, \mathcal{F}_{R_w}) \leq n. \] (2.30)

Although the left-hand side of (2.30) depends on \( m \), the inequality remains true for all \( m \). The quantity \( \epsilon^*_n \) is an upper bound on the entropy number of \( \mathcal{F}_{R_w} \), which is the functional inverse of the covering number. In this theorem, the number \( j^*_n \) has a natural interpretation: If \( j^*_n = d \) is independent of \( n \), then from (2.29) we can obtain
\[ \epsilon_n(\mathcal{F}_{R_w}) = O \left( \left( \frac{1}{n} \right)^{\frac{1}{d}} \right) \Rightarrow \sup_{X^m \in X^m} \mathcal{N}(\epsilon, \mathcal{F}_{R_w}, \ell_X^m) = O \left( \left( \frac{1}{\epsilon} \right)^d \right). \]

Hence, for a given value of \( n \), \( j^*_n \) can be viewed as the effective dimension of the function class. Clearly, this effective dimension depends on the rate of decay of the eigenvalues. As expected, for smooth kernels (which have rapidly decreasing eigenvalues), the effective dimension is small. In the following, we write \( j^* \) for \( j^*_n \).

### 2.2.3 Proof outline

The proof of Theorem 2.8 is quite long and is in Section 2.6. It involves the following four steps.

1. We first prove that for all \( n \in \mathbb{N} \),
\[ \hat{j} = \min \left\{ j : \lambda_{j+1} \left( \frac{\lambda_1 \cdots \lambda_j}{n^2} \right)^{\frac{1}{j}} \right\} \] (2.31)
exists, whenever \( (\lambda_i)_i \) are the eigenvalues of a Mercer kernel.

2. We then prove that for any \( n \in \mathbb{N} \)
\[ \inf_{(a_s)_s} \sup_{(\sqrt{\lambda_i}/a_s)_s} \inf_{j \in \mathbb{N}} \inf_{(a_s)_s \in A_j} B((a_s)_s, n, j) \leq \inf_{j \in \mathbb{N}} \inf_{(a_s)_s \in A_j} B((a_s)_s, n, j). \] (2.32)

3. The next step is to prove that the choice of \( (a_s)_s \) and \( j \) described by (2.25) and (2.26) are optimal. It is separated into two parts:

   (a) For any \( j_0 \leq j^* \), and any \( (a_s)_s \in A_{j_0} \),
   \[ B((a_s)_s, n, j_0) \geq B((a_s^*_s)_s, n, j^*) \]
   holds.
(b) For any $j_0 > j^*$, and any $(a_s)_s \in A_{j_0}$,

$$B((a_s)_s, n, j_0) \geq B((a^*_s)_s, n, j^*)$$

also holds.

4. Finally we show that $(a^*_s)_s \in A_j$ and $(\sqrt{\lambda_s}/a^*_s)_s \in \ell_2$ when $(a^*_s)_s$ is chosen according to (2.26).

### 2.3 Example

We illustrate the results of this chapter with an example. Consider the kernel

$$k(x, y) = k(x - y), \text{ where } k(x) = \exp -\frac{x^2}{\sigma^2}.$$ 

Here, $d = 1$. For such kernels (RBF kernels), $\|\Psi(x)\|_{\ell_2} = 1$ for all $x \in X$. Thus by Mercer’s theorem the class (2.6) can be written as

$$\mathcal{F}_{Rw} = \{ x \mapsto \langle w, \tilde{x} \rangle : \tilde{x} \in \mathcal{S}, \|\tilde{x}\|_{\ell_2} \leq 1, \|w\|_{\ell_2} \leq R_w \}.$$ 

(See, for example, [16] for a more detailed explanation of this point — it is the fundamental basis of viewing support vector machines in feature space.) One can use the fat-shattering dimension to bound the covering number of the class of functions $\mathcal{F}_{Rw}$ (see, for example, [2]).

**Theorem 2.11** With $\mathcal{F}_{Rw}$ as above, if $m \geq 16R^2_w/\epsilon^2 \geq 1$,

$$\log \mathcal{N}^m(\epsilon, \mathcal{F}_{Rw}) \leq 48 \left( \frac{R_w}{\epsilon} \right)^2 \log^2 \left( \frac{4eR_w m}{\epsilon} \right). \quad (2.33)$$

In order to determine the eigenvalues of $T_k$, we need to periodize the kernel. This periodization is necessary in order to get a discrete set of eigenvalues since $k(x)$ has infinite support (see [60] for further details). For our purposes, we can assume a fixed period $2\pi/\omega_0$ for some $\omega_0 > 0$. Since the kernel is translation invariant, the eigenfunctions are

$$\psi_n(x) = \sqrt{2} \cos(n\omega_0 x)$$
and so $C_k = \sqrt{2}$ ([60]). The $\sqrt{2}$ factor comes from the requirement in Theorem 2.1 that $\|\psi_j\|_{\ell_2} = 1$. The eigenvalues can be computed and are

$$\lambda_j = \sqrt{2\pi \sigma} e^{-\frac{\omega_0^2}{4} \sigma^2 j^2}.$$ 

Setting $c_1 = \sqrt{2\pi \sigma}$, $c_2 = \frac{\omega_0^2}{4} \sigma^2$, the eigenvalues can be written as

$$\lambda_j = c_1 e^{-c_2 j^2}. \quad (2.34)$$

From (2.25), we know that $\lambda_{j+1} < \left(\frac{\lambda_1 \cdots \lambda_j}{n^2}\right)^{\frac{1}{j}}$ implies $j^* \leq j$. But (2.34) shows that this condition on the eigenvalues is equivalent to

$$c_1 e^{-c_2 (j+1)^2} < n^{-\frac{1}{j}} \left(c_1 e^{-c_2 \sum_{i=1}^{j} i^2}\right)^{\frac{1}{j}}, \quad (2.35)$$

which is equivalent to

$$c_2 (j+1)^2 > \frac{2}{j} \ln n + \frac{c_2}{6} (j+1)(2j+1) \Leftrightarrow \frac{2}{3} c_2 (j+1) j \left(j + \frac{5}{4}\right) > 2 \ln n,$$

which follows from

$$j > \left(\frac{12 \ln n}{\omega_0^2 \sigma^2}\right)^{1/3}.$$ 

Hence,

$$j^* \leq \left[\left(\frac{12 \ln n}{\omega_0^2 \sigma^2}\right)^{1/3}\right] + 1. \quad (2.36)$$

We can now use (2.29) to give an upper bound on $\epsilon_n$. Since the $\lambda_i$ decay so rapidly the tail $\sum_{i=j^*+1}^{\infty} \lambda_i$ in (2.29) is dominated by the first term. We obtain the following bound.

$$\epsilon_n^2 = O \left(j^* n^{-\frac{3}{j^*}} c_1 \exp \left(-\frac{c_2}{6} (j^* + 1)(2j^* + 1)\right)\right).$$

Substituting (2.36) shows that

$$\log \epsilon_n = O \left(\log \log n + \log \sigma - (\sigma \log n)^{\frac{3}{2}}\right) \quad (2.37)$$

We can get several results from Equation (2.37).
The relationship between $\epsilon_n$ and $n$. For fixed $\sigma$, (2.37) shows that

$$\log 1/\epsilon_n = \Omega(\log^{3/2} n),$$

which implies

$$\log \mathcal{N}^m(\epsilon, \mathcal{F}_{R_w}) = O\left(\log^{3/2} \left(\frac{1}{\epsilon}\right)\right), \quad (2.38)$$

which is considerably better than Theorem 2.11. This can also be seen in Figure 2.3. Note that (2.38) does not depend on $m$. This is a consequence of using (2.22) which also has no dependence on $m$. One can obtain a dependence in $m$ if instead of (2.22) one uses equation (49) of [60]. As explained in [60], for moderate decay rates of $(\lambda_i)$, the bounds obtained are no better by doing so.

The relationship between $\epsilon_n$ and $\sigma^2$. Here, $\sigma^2/2$ is the variance of the gaussian functions. When $\sigma^2$ increases, the kernel function will be wider, so the class $\mathcal{F}_{R_w}$ should be simpler. In Equation (2.37), we notice that if $\sigma$ decreases, $\epsilon_n$ decreases for fixed $n$. Similarly, if $\sigma$ increases, $n$ decreases for fixed $\epsilon_n$. Since the entropy numbers (and the covering numbers) indicate the capacity of the learning machine, the more complicated the machine is, the bigger the covering
numbers are for fixed $\epsilon_n$. Specifically we see from Equation (2.37) that
\[
\log 1/\epsilon_n = \Omega(1/\sigma^3),
\]
and that
\[
\log N^m(\epsilon, \mathcal{F}_{R_w}) = O(1/\sigma). \tag{2.39}
\]

Figures 2.4 and 2.5 illustrate the bounds on the effective dimension $j^*$ (for $\sigma^2 = 1$) as a function of $n$ and $\epsilon$, respectively.

### 2.4 Experiment

This section describes experiments that were carried out to test the predictive accuracy of the results obtained in Section 2.3. The generalization error bound (1.10) from Theorem 1.11 is of use.
2.4. EXPERIMENT

Figure 2.5: $j^*$ versus $\epsilon$ for a gaussian kernel. Since $j^*$ can be interpreted as an "effective dimension", this clearly illustrates why the bound on the covering numbers for gaussian kernels grows so slowly as $\epsilon \downarrow 0$. Even when $\epsilon = 10^{-9}$, $j^*$ is only 13.

Inequality (1.10) shows that for all $m, \delta \in (0, 1)$, and $P(x, y)$, with probability at least $1 - \delta$ over $S^m$ chosen according to $P^m$,

$$R_P[f] - R_{\text{emp}}^\gamma[f, S^m] \leq \sqrt{\frac{8}{m} \ln \left( \frac{2N^{2m}(\gamma/2, \mathcal{F})}{\delta} \right)}.$$

Thus we get

$$\sup_{f \in \mathcal{F}} (R_P[f] - R_{\text{emp}}^\gamma[f, S^m]) = O \left( \ln^{\frac{1}{2}} (N^{2m}(\gamma/2, \mathcal{F})) \right) \quad (2.40)$$

The experiment was a simple binary classification problem. The input space was the ball $B = \{x \in \mathbb{R}^2: \|x\|_2 \leq 1\}$. The true classification function to be learned was 1 when the points were in the first and third quadrants, and 0 otherwise. For each trial a training sample $S = \{(x_i, y_i)\}_{i=1}^{50}$ of size 50 was generated with the $x_i$ drawn independently at random from a uniform distribution on $B$. Independent label noise of 10% was applied to the labels $y_i$. A support vector machine using a gaussian radial basis function kernel with width $\sigma$ was trained on $S$. The regularization parameter
Combining (2.39) and (2.40) we obtain
\[
\log \left( \sup_{f \in \mathcal{F}} (R_P[f] - R_{\text{emp}}^\gamma[f, S^m]) \right) = O \left( \log \left( \frac{1}{\sigma} \right) \right). \tag{2.41}
\]

Thus we expect that if we fix the value of \( \gamma \) (for the sake of computing \( R_{\text{emp}}^\gamma[f, S^m] \)) and plot the log of the test error versus the log of \( \sigma \) we should get a straight line.

Figure 2.6 shows the log of the difference between the test error and the \( \gamma \)-margin empirical error for a range of different \( \gamma \) (top curve \( \gamma = 0 \), bottom curve \( \gamma = 0.25 \)). The values are plotted against \( \log(\sigma) \). As expected the curves are close to straight lines.
2.5 Conclusions

We have presented a new formula for bounding the covering numbers of support vector machines in terms of the eigenvalues of an integral operator induced by the kernel. We showed, by way of an example using a gaussian kernel, that the new bound is easily computed and considerably better than previous results that did not take account of the kernel. We showed explicitly the effect of the choice of width of the kernel in this case.

The "effective dimension", \( j^* \), can illustrate the character of kernel expansions clearly. For a smooth kernel, the "effective dimension" \( j^* \) is small. The value of \( j^* \) depends on \( n \) which in turn depends on \( \epsilon \). Thus \( j^* \) can be considered analogous to existing "scale-sensitive" dimensions, such as the fat-shattering dimension. A key difference is that we now have bounds for \( j^* \) that explicitly depend on the kernel.

The bounds obtained apply to any dimension \( d \). However, repeated eigenvalues become generic for isotropic translation invariant kernels. It is possible to obtain bounds that can be tighter in some cases, by using a slightly more refined argument; see [60].

2.6 Proof of Theorem 2.10

Step one

As indicated in Section 2.2.1, we will first prove the existence of \( \hat{j} \), which is defined in (2.31).

**Lemma 2.12** Suppose \( \lambda_1 \geq \lambda_2 \geq \cdots \geq 0 \) is a non-increasing sequence of non-negative numbers and \( \lim_{j \to \infty} \lambda_j = 0 \). Then for all \( n \in \mathbb{N} \), there exists \( \hat{j} \in \mathbb{N} \) such that

\[
\lambda_{\hat{j}+1} < \left( \frac{\lambda_1 \cdots \lambda_{\frac{1}{2}}} {n^2} \right)^{\frac{1}{\hat{j}}}. \tag{2.42}
\]
Proof Let \( P_j = \frac{\lambda^*_j}{\lambda_1 \cdots \lambda_j} \). Observe that (2.42) can be written as \( P_j < \frac{1}{n^2} \), and hence for all \( n \) there is a \( j \) such that (2.42) is true iff \( \lim_{j \to \infty} P_j = 0 \). But

\[
P_j = \frac{\lambda^*_j}{\lambda_1 \cdots \lambda_j} = \frac{\lambda^*_j}{\lambda_j} \prod_{i=2}^{j} \frac{\lambda^*_j}{\lambda_i} \leq \frac{\lambda^*_j}{\lambda_1}
\]

since \((\lambda_i)\) is non-increasing. Since \( \lim_{j \to \infty} \lambda_j = 0 \), we get \( \lim_{j \to \infty} P_j = 0 \). Thus for any \( n \in \mathbb{N} \) there is a \( j \) such that (2.42) is true. \( \square \)

Corollary 2.13 Suppose \( k \) is a Mercer kernel and \( T_k \) the associated integral operator. If \( \lambda_i = \lambda_i(T_k) \), then the minimum \( j \) from (2.31) always exists.

Proof By Mercer's Theorem, \( (\lambda_i) \in \ell_1 \) and so \( \lim_{i \to \infty} \lambda_i = 0 \). Lemma 2.12 can thus be applied. \( \square \)

Step two

Lemma 2.14 Suppose \( A_j \) and \( B((a_s)_s, n, j) \) are defined as in (2.23) and (2.24), \((\sqrt{A_s}/a^*_s)_s \in \ell_2, j^* \) and \((a^*_s)_s \in A_{j^*} \) satisfy

\[
B((a^*_s)_s, n, j^*) = \inf_{j \in \mathbb{N}} \inf_{(a_s)_s \in A_j} B((a_s)_s, n, j).
\]

Then

\[
\inf_{(a_s)_s, (\sqrt{A_s}/a^*_s)_s \in \ell_2} \sup_{n, j \in \mathbb{N}} B((a_s)_s, n, j) \leq \inf_{j \in \mathbb{N}} \inf_{(a_s)_s \in A_j} B((a_s)_s, n, j) \leq B((a^*_s)_s, n, j^*).
\]

Proof Since \((\sqrt{A_s}/a^*_s)_s \in \ell_2,\)

\[
\inf_{(a_s)_s, (\sqrt{A_s}/a^*_s)_s \in \ell_2} \sup_{n, j \in \mathbb{N}} B((a_s)_s, n, j) \leq \sup_{j \in \mathbb{N}} B((a^*_s)_s, n, j). \tag{2.45}
\]

But \((a^*_s)_s \in A_{j^*} \), following the definition of \( A_j \) and equality (2.43) we get

\[
\sup_{j \in \mathbb{N}} B((a^*_s)_s, n, j) = B((a^*_s)_s, n, j^*) = \inf_{j \in \mathbb{N}} \inf_{(a_s)_s \in A_j} B((a_s)_s, n, j).
\]
In fact, we can show that Inequality (2.44) is an equality. The proof is in Section 2.7.

It is now easier to calculate the optimal bound of the entropy number using Lemma 2.14.

**Step three**

In this step, we will prove that the choice of \((a^*_s)_s\) and \(j^*\) given in Theorem 2.8 are optimal. We will first prove a useful technical result.

**Lemma 2.15** Suppose \(A_j\) and \((\lambda_i)_i\) are defined as above, \((a_s)_s \in A_{j_0}\).

Then we have

\[
\left( \sum_{i=j_0+1}^{\infty} \frac{\lambda_i}{a_i^2} \right) \left( \frac{a_1 \cdots a_{j_0}}{n} \right)^{\frac{1}{j_0}} - \sum_{i=j_0+1}^{\infty} \lambda_i \geq 0. \tag{2.46}
\]

**Proof** Since \((a_s)_s \in A_{j_0}\), the following inequality must be true for \(k \in \mathbb{N}:

\[
\left( \frac{a_1 \cdots a_{j_0} \cdots a_{j_0+k}}{n} \right)^{\frac{1}{j_0+k}} \leq \left( \frac{a_1 \cdots a_{j_0}}{n} \right)^{\frac{1}{j_0}}. \tag{2.47}
\]

which implies

\[
\left( \frac{a_1 \cdots a_{j_0} \cdots a_{j_0+k}}{n} \right) \leq \left( \frac{a_1 \cdots a_{j_0}}{n} \right)^{\frac{k}{j_0}}, \quad \forall k \in \mathbb{N}. \tag{2.48}
\]

Set

\[\psi = \left( \frac{a_1 \cdots a_{j_0}}{n} \right)^{\frac{1}{j_0}}.\]

Then (2.48) can be rewritten as:

\[a_{j_0+1} \cdots a_{j_0+k} \leq \psi^k, \quad \forall k \in \mathbb{N}. \tag{2.49}\]

Hence, the left hand side of (2.46) can be rewritten as

\[
\sum_{i=j_0+1}^{\infty} \frac{\lambda_i}{a_i^2} \psi^2 - \sum_{i=j_0+1}^{\infty} \lambda_i \psi^2 \sum_{i=j_0+1}^{\infty} \lambda_i \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right). \tag{2.50}
\]
From (2.49), we get \( a_{j_0 + 1} \leq \psi \), so
\[
\frac{1}{a_{j_0 + 1}^2} - \frac{1}{\psi^2} \geq 0.
\]

Suppose \( \frac{1}{a_i^2} - \frac{1}{\psi^2} < 0 \) for some \( i \in \mathbb{N} \). We will separate the sum into several parts. Set
\[
\begin{align*}
  k_0 &= j_0, \\
  k_m &= \max\{n > l_m : \frac{1}{a_i^2} < \frac{1}{\psi^2}, \forall i \in \{l_m + 1, \ldots, n\}\}, \\
  l_m &= \max\{n > k_m - 1 : \frac{1}{a_i^2} \geq \frac{1}{\psi^2}, \forall i \in \{k_m - 1 + 1, \ldots, n\}\},
\end{align*}
\]

where we set \( k_m \) and \( l_m \) to \( \infty \) if the max does not exist. Since \( (\lambda_i)_i \) is a non-increasing sequence, from (2.51) we know
\[
\lambda_i \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) \geq \lambda_{i+c} \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right)
\]
\( \forall i \in \{k_m - 1 + 1, \ldots, l_m\}, c \in \mathbb{N} \)
\[
\lambda_i \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) > \lambda_{i-c} \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right)
\]
\( \forall i \in \{l_m + 1, \ldots, k_m\}, \forall c \in \{1, \ldots, i-1\} \)

for \( m \in \mathbb{N} \). Hence, if \( l_m \) is finite,
\[
\sum_{i=k_m-1+1}^{k_m} \lambda_i \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) \\
\geq \lambda_{l_m} \sum_{i=k_m-1+1}^{l_m} \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) + \lambda_{l_m} \sum_{i=l_m+1}^{k_m} \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) \\
= \lambda_{l_m} \sum_{i=k_m-1+1}^{k_m} \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right). 
\]

And if \( l_m \) is infinite, this inequality is clearly true. We will exploit the inequality of the arithmetic and geometric means
\[
x_1 + x_2 + \ldots + x_m \geq m(x_1 \cdots x_m)^{\frac{1}{m}} \text{ for } x_i > 0.
\]

Now (2.53) implies that for any \( k_0 + 1 \leq j \leq k_m \), we have
\[
\sum_{i=k_0+1}^{j} \frac{1}{a_i^2} \geq (j - k_0) \left( \prod_{i=k_0+1}^{j} \frac{1}{a_i^2} \right)^{\frac{1}{j-k_0}}. 
\]
which together with (2.49) gives

\[ \sum_{i=k_0+1}^{j} \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) = \sum_{i=k_0+1}^{j} \frac{1}{a_i^2} - \frac{j - k_0}{\psi^2} \geq 0. \]  \hspace{1cm} (2.55)

Hence, for any \( k_m \), finite or infinite,

\[ \sum_{i=k_0+1}^{k_m} \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) \geq 0. \]  \hspace{1cm} (2.56)

Now, for all \( k_m \), using (2.52) and (2.56) repeatedly, we get

\[
\sum_{i=k_0+1}^{k_m} \lambda_i \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) \\
= \sum_{i=k_0+1}^{k_1} \lambda_i \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) + \cdots + \sum_{i=k_{m-1}+1}^{k_m} \lambda_i \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) \\
\geq \lambda_{i_1} \sum_{i=k_0+1}^{k_1} \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) + \cdots + \lambda_{i_m} \sum_{i=k_{m-1}+1}^{k_m} \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) \\
\geq \cdots \geq \lambda_{i_m} \sum_{i=k_0+1}^{k_m} \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) \geq 0.
\]

for all \( m \in \mathbb{N} \). Hence

\[ \psi^2 \sum_{i=j_0+1}^{\infty} \lambda_i \left( \frac{1}{a_i^2} - \frac{1}{\psi^2} \right) \geq 0. \]  \hspace{1cm} (2.57)

Noticing (2.50), inequality (2.46) is true.

Now, let us prove the main result.

**Lemma 2.16** Let \( A_j \) and \( B((a_s)_s, n, j) \) be defined as above. Then we have

\[ B((a_s)_s, n, j^*) = \inf_{j_0 \in \mathbb{N}} \inf_{(a_s)_s \in A_{j_0}} B((a_s)_s, n, j_0), \]  \hspace{1cm} (2.58)

where

\[ a_i^* = \begin{cases} \sqrt{\lambda_i} & \text{when } i \leq j^* \\ \left( \sqrt[3]{\lambda_1 \cdots \lambda_j^*} \right)^{\frac{1}{2}} & \text{when } i > j^* \end{cases} \]  \hspace{1cm} (2.59)

\[ j^* = \min \left\{ j : \lambda_{j+1} < \left( \frac{\lambda_1 \cdots \lambda_j}{n^2} \right)^{\frac{1}{2}} \right\}. \]  \hspace{1cm} (2.60)
**Proof** The main idea is to compare $B^2((a_s)s, n, j_0)$ with $B^2((a^*_s)s, n, j^*)$ and show

$$B^2((a_s)s, n, j_0) \geq B^2((a^*_s)s, n, j^*)$$

for all $j_0 \in \mathbb{N}$ and any $(a_s)s \in A_{j_0}$. From the definition of $B((a_s)s, n, j)$, we know

$$B^2((a_s)s, n, j_0) = \left( \sum_{i=1}^{\infty} \frac{\lambda_i}{a_i^2} \right) \left( \frac{a_1 \cdots a_{j_0}}{n} \right)^{\frac{2}{j_0}}$$

and

$$B^2((a^*_s)s, n, j^*) = j^* \left( \frac{\lambda_1 \cdots \lambda_{j^*}}{n^2} \right)^{\frac{1}{j^*}} + \sum_{i=j^*+1}^{\infty} \lambda_i.$$ 

For convenience, we set

$$\Lambda = \left( \frac{\lambda_1 \cdots \lambda_{j^*}}{n^2} \right)^{\frac{1}{j^*}}.$$ 

Hence,

$$B^2((a_s)s, n, j_0) - B^2((a^*_s)s, n, j^*) = \sum_{i=1}^{\infty} \frac{\lambda_i}{a_i^2} \left( \frac{a_1 \cdots a_{j_0}}{n} \right)^{\frac{2}{j_0}} - j^* \Lambda - \sum_{i=j^*+1}^{\infty} \lambda_i. \quad (2.61)$$

**Part a:** for the condition $j_0 \leq j^*$.

Rewrite (2.61):

$$B^2((a_s)s, n, j_0) - B^2((a^*_s)s, n, j^*) = \left\{ \sum_{i=1}^{j_0} \frac{\lambda_i}{a_i^2} \left( \frac{a_1 \cdots a_{j_0}}{n} \right)^{\frac{2}{j_0}} - j_0 \left( \frac{\lambda_1 \cdots \lambda_{j_0}}{n^2} \right)^{\frac{1}{j_0}} \right\}$$

$$- \left( \sum_{i=j_0+1}^{\infty} \lambda_i \right)$$

$$- \left( j^* \Lambda + \sum_{i=j^*+1}^{\infty} \lambda_i - j_0 \left( \frac{\lambda_1 \cdots \lambda_{j_0}}{n^2} \right)^{\frac{1}{j_0}} - \sum_{i=j_0+1}^{\infty} \lambda_i \right)$$

$$= \left\{ \sum_{i=1}^{j_0} \frac{\lambda_i}{a_i^2} \left( \frac{a_1 \cdots a_{j_0}}{n} \right)^{\frac{2}{j_0}} - j_0 \left( \frac{\lambda_1 \cdots \lambda_{j_0}}{n^2} \right)^{\frac{1}{j_0}} \right\}$$

$$+ \left\{ \sum_{i=j_0+1}^{\infty} \frac{\lambda_i}{a_i^2} \left( \frac{a_1 \cdots a_{j_0}}{n} \right)^{\frac{2}{j_0}} - \sum_{i=j_0+1}^{\infty} \lambda_i \right\}$$
2.6. PROOF OF THEOREM 2.10

\[ 2.62 \quad E_1 = E_1 + E_2 + E_3. \]

We will show \( E_1 \geq 0, E_2 \geq 0 \) and \( E_3 \geq 0 \).

To prove \( E_1 \geq 0 \).

Since \( \lambda_i \geq 0 \) and \( a_i \geq 0 \), we exploit the inequality of the arithmetic and geometric means (2.53) again. Hence

\[ E_1 \geq j_0 \left( \frac{\lambda_1 \cdots \lambda_{j_0}}{n^2} \right) \left( \frac{1}{j_0} \right) + j_0 \left( \frac{1}{j_0} \right) = 0 \]

To prove \( E_2 \geq 0 \).

Applying Lemma 2.15 shows \( E_2 \geq 0 \).

To prove \( E_3 \geq 0 \).

In order to prove \( E_3 \geq 0 \), let us define the function

\[ g(j) = j \left( \frac{\lambda_1 \cdots \lambda_j}{n^2} \right)^{\frac{1}{j}} + \sum_{i=j+1}^{\infty} \lambda_i. \]

We will show that \( g(j) \) is a non-increasing function of \( j \), for \( j \leq j^* \). Set

\[ \beta_j = \left( \frac{\lambda_1 \cdots \lambda_j}{n^2} \right)^{\frac{1}{j}}, \quad \beta_{j-1} = \left( \frac{\lambda_1 \cdots \lambda_{j-1}}{n^2} \right)^{\frac{1}{j-1}}, \]

we have

\[ g(j - 1) - g(j) = (j - 1)\beta_{j-1} + \lambda_j - j\beta_j \]

\[ = (\lambda_j - \beta_{j-1}) - j(\beta_j - \beta_{j-1}). \]
Noticing \( \beta_{j-1}^{j} \lambda_j = \beta_j^j \), (2.65) can be modified to
\[
g(j - 1) - g(j) = \beta_{j-1}^{j-1} \left( (\beta_j^j - \beta_j^{j-1}) - j \beta_{j-1}^{j-1} (\beta_j - \beta_{j-1}) \right). \tag{2.66}
\]
Since \( j \leq j^* \), following (2.60), we get
\[
\lambda_j \geq \left( \frac{\lambda_1 \cdots \lambda_{j-1}}{n^2} \right)^{\frac{1}{j-1}} \quad \forall j \leq j^*. \tag{2.67}
\]
So
\[
\beta_j = \left( \frac{\lambda_1 \cdots \lambda_{j-1}}{n^2} \right)^{\frac{j}{2}} \lambda_j^\frac{1}{2} \geq \left( \frac{\lambda_1 \cdots \lambda_{j-1}}{n^2} \right)^{\frac{j}{2}} \geq 0.
\]
Making use of the formula
\[
x^n - y^n = (x - y) \sum_{i=1}^{n} x^{n-i} y^{i-1}, \tag{2.68}
\]
we obtain
\[
\beta_j^j - \beta_{j-1}^{j-1} = (\beta_j - \beta_{j-1}) \sum_{i=1}^{j} \beta_j^{j-i} \beta_{j-1}^{i-1} \geq \beta_{j-1}^{j-1} (\beta_j - \beta_{j-1}).
\]
Together with \( \beta_{j-1} > 0 \) and (2.66), we obtain
\[
(\lambda_j - \beta_{j-1}) - j (\beta_j - \beta_{j-1}) \geq 0.
\]
Hence,
\[
g(j - 1) \geq g(j).
\]
Since \( j_0 \leq j^* \), we get
\[
E_3 = g(j_0) - g(j^*) \geq 0. \tag{2.69}
\]
Combining the above results, we get
\[
B^2((a_s)_s, n, j_0) - B^2((a_s^*)_s, n, j^*) \geq 0 \quad \forall j_0 \leq j^*. \tag{2.70}
\]
2.6. PROOF OF THEOREM 2.10

Part b: for the condition $j_0 > j^*$.

Rewrite (2.61):

\[
B^2((a_s)_s, n, j_0) - B^2((a^*_s)_s, n, j^*)
= \left( \sum_{i=1}^{j_0} \frac{\lambda_i}{a_i^2} + \sum_{i=j_0+1}^{\infty} \frac{\lambda_i}{a_i^2} \right) \left( \frac{a_1 \cdots a_{j_0}}{n} \right)^{\frac{2}{j_0}} \nonumber
- j^* \Lambda - \sum_{i=j^*+1}^{j_0} \lambda_i - \sum_{i=j_0+1}^{\infty} \lambda_i
= \left\{ \sum_{i=1}^{j_0} \frac{\lambda_i}{a_i^2} \left( \frac{a_1 \cdots a_{j_0}}{n} \right)^{\frac{2}{j_0}} - j^* \Lambda - \sum_{i=j_0+1}^{\infty} \lambda_i \right\}
+ \left\{ \sum_{i=j_0+1}^{\infty} \frac{\lambda_i}{a_i^2} \left( \frac{a_1 \cdots a_{j_0}}{n} \right)^{\frac{2}{j_0}} - \sum_{i=j_0+1}^{\infty} \lambda_i \right\}
= F_1 + F_2. \tag{2.71}
\]

We will show $F_1 \geq 0$ and $F_2 \geq 0$.

To prove $F_1 \geq 0$.

For convenience, we set

\[ D_i = \left( \frac{a_1 \cdots a_i}{n} \right)^{\frac{2}{i}}. \]

$F_1$ can be rewritten as:

\[
\left( \sum_{i=1}^{j^*} \frac{\lambda_i}{a_i^2} + \sum_{i=j^*+1}^{j_0} \frac{\lambda_i}{a_i^2} \right) D_{j_0} - j^* \Lambda - \sum_{i=j^*+1}^{j_0} \lambda_i
= \left\{ D_{j_0} \sum_{i=1}^{j^*} \frac{\lambda_i}{a_i^2} - j^* \Lambda \right\}
+ \left\{ \sum_{i=j^*+1}^{j_0} \lambda_i \left( \frac{D_{j_0}}{a_i^2} - 1 \right) \right\}
= P_1 + P_2. \tag{2.72}
\]

Let us consider $P_1$ at first.

\[
P_1 = (D_{j_0} + D_{j^*} - D_{j^*}) \sum_{i=1}^{j^*} \frac{\lambda_i}{a_i^2} - j^* \Lambda
= D_{j^*} \sum_{i=1}^{j^*} \frac{\lambda_i}{a_i^2} - j^* \Lambda + (D_{j_0} - D_{j^*}) \sum_{i=1}^{j^*} \frac{\lambda_i}{a_i^2}.
\]
Since \((\lambda_i/a_i^2) > 0\), using the inequality of the arithmetic and geometric mean (2.53) again, we get
\[
\sum_{i=1}^{j^*} \lambda_i a_i^{-2} \geq j^* \left( \frac{\lambda_1 \cdots \lambda_{j^*}}{a_1^2 \cdots a_{j^*}^2} \right) \frac{j^*}{n^2} = \frac{j^*}{D_{j^*}}. 
\]
Since \((a_s)_s \in A_{j_0}\), we get \(D_{j_0} \geq D_i\) for any \(i \neq j_0\) and
\[
\lambda_{j^*+1} < \left( \frac{\lambda_1 \cdots \lambda_{j^*}}{n^2} \right)^{\frac{1}{j^*}}
\]
holds based on (2.60). Hence
\[
P_1 \geq 0 + (D_{j_0} - D_{j^*}) \sum_{i=1}^{j^*} \frac{\lambda_i}{a_i^2} \geq (D_{j_0} - D_{j^*}) \frac{1}{D_{j^*}} j^* A \\
> (D_{j_0} - D_{j^*}) \frac{1}{D_{j^*}} j^* A_{j^*+1} \geq 0. 
\]
(2.73)

Let us consider \(P_2\) now. If \(P_2 \geq 0\), then \(F_1 \geq 0\).

So let us prove that \(F_1 \geq 0\) is also true when \(P_2 \leq 0\). Observing \(a_i^2 = D_i/D_{i-1}\) and \(D_{j_0} \geq D_i\) for any \(i \neq j_0\), the last element of \(P_2\)
\[
\lambda_{j_0} \left( \frac{D_{j_0}}{a_{j_0}^2} - 1 \right) = \lambda_{j_0} \left( \left( \frac{D_{j_0-1}}{D_{j_0}} \right)^{j_0-1} - 1 \right) \leq 0.
\]

Using the similar method as before. Suppose \(\frac{D_{j_0}}{a_i^2} - 1 > 0\) for some \(i \in (j^*, j_0)\). We separate \(P_2\) into several parts. Set
\[
k_0 = j_0 + 1, \\
l_m = \min\{n < k_m : \frac{D_{j_0}}{a_i^2} - 1 \leq 0, \forall i \in \{n, \cdots, k_m - 1\}\}, \\
k_m = \min\{n < l_m-1 : \frac{D_{j_0}}{a_i^2} - 1 > 0, \forall i \in \{n, \cdots, l_m-1\}\}.
\]
(2.74)

Since \((\lambda_i)_i\) is a non-increasing sequence, from (2.74) we know
\[
\lambda_i \left( \frac{D_{j_0}}{a_i^2} - 1 \right) \geq \lambda_{i+c} \left( \frac{D_{j_0}}{a_i^2} - 1 \right) \\
\forall i \in \{k_{m+1}, \cdots, l_m - 1\}, c \in \mathbb{N}
\]
\[
\lambda_i \left( \frac{D_{j_0}}{a_i^2} - 1 \right) > \lambda_{i-c} \left( \frac{D_{j_0}}{a_i^2} - 1 \right) \\
\forall i \in \{l_m, \cdots, k_m - 1\}, \forall c \in \{1, \cdots, i - 1\}. 
\]
(2.75)
Using (2.75), we have
\[ \sum_{i=k_{m+1}}^{k_{m-1}} \lambda_i \left( \frac{D_{i,0}}{a_i^2} - 1 \right) \]
\[ \geq \lambda_{l_0} \sum_{i=k_{m+1}}^{l_{m-1}} \left( \frac{D_{i,0}}{a_i^2} - 1 \right) + \lambda_{l_m} \sum_{i=l_m}^{k_{m-1}} \left( \frac{D_{i,0}}{a_i^2} - 1 \right) \]
\[ = \lambda_{l_m} \sum_{i=k_{m+1}}^{k_{m-1}} \left( \frac{D_{i,0}}{a_i^2} - 1 \right). \] (2.76)

Hence,
\[ 0 \geq P_2 = \sum_{i=j^*+1}^{j_0} \left( \frac{D_{i,0}}{a_i^2} - 1 \right) \lambda_i \]
\[ = \sum_{i=j^*+1}^{k_1-1} \left( \frac{D_{i,0}}{a_i^2} - 1 \right) \lambda_i + \sum_{i=k_1}^{k_{0-1}} \left( \frac{D_{i,0}}{a_i^2} - 1 \right) \lambda_i \]
\[ \geq \sum_{i=j^*+1}^{k_{0-1}} \left( \frac{D_{i,0}}{a_i^2} - 1 \right) \lambda_i + \lambda_{l_2} \sum_{i=k_1}^{k_{0-1}} \left( \frac{D_{i,0}}{a_i^2} - 1 \right). \]

If \( \lambda_1 \sum_{i=k_1}^{k_{0-1}} \left( \frac{D_{i,0}}{a_i^2} - 1 \right) > 0 \), we get
\[ P_2 > \sum_{i=j^*+1}^{k_{0-1}} \left( \frac{D_{i,0}}{a_i^2} - 1 \right) \lambda_i. \]

If \( \lambda_1 \sum_{i=k_1}^{k_{0-1}} \left( \frac{D_{i,0}}{a_i^2} - 1 \right) \leq 0 \), we can use (2.75) and (2.76) repeatedly. Finally, using (2.53) and \( a_i^2 = D_{i,1}/D_{i,1}^{-1} \) again, we can get
\[ 0 \geq P_2 \geq \sum_{i=j^*+1}^{j^*+l} \left( \frac{D_{i,0}}{a_i^2} - 1 \right) \lambda_{j^*+l} = \lambda_{j^*+l} \sum_{i=j^*+1}^{j^*+l} \left( \frac{D_{i,0}}{a_i^2} - 1 \right) \]
\[ \geq \lambda_{j^*+l} \left( \left. \frac{1}{a_{j^*+1} \cdots a_{j^*+l}} \right. \right) \frac{1}{D_{j,0}} \]
\[ = \lambda_{j^*+l} \left( \frac{D_{j,0} \left( D_{j^*+l}^* \right)^{-\frac{l}{l}} - 1}{D_{j^*+l}} \geq \lambda_{j^*+l} \left( \frac{D_{j^*+l}^*}{D_{j^*+l}} \right) \right) \]
\[ \geq \lambda_{j^*+l} \left( \frac{D_{j^*+l}^*}{D_{j,0}} - 1 \right) \text{ with } l \in \{1, \cdots, k_{0-1}\}. \] (2.77)

Combining (2.73) and (2.77), we have
\[ F_1 = P_1 + P_2 > \frac{j^* \lambda_{j^*+l} (D_{j,0} - D_{j^*})}{D_{j^*}} + \lambda_{j^*+l} \left( \frac{D_{j^*}{j^*}}{D_{j,0}} - 1 \right). \] (2.78)
In order to show $F_1 \geq 0$, we just need to show

$$j^* \lambda_{j^*+1} \left( D_{j^*} - D_{j^*+1} \right) + \lambda_{j^*+l} \left( \frac{D_{j^*+l}}{D_{j^*}} \right) \geq 0. \quad (2.79)$$

When $D_{j^*} = D_{j_0}$,

$$j^* \lambda_{j^*+1} \left( D_{j_0} - D_{j^*+1} \right) + \lambda_{j^*+l} \left( \frac{D_{j^*+l}}{D_{j_0}} \right) = 0.$$

Inequality (2.79) holds.

When $D_{j_0} > D_{j^*}$, setting $\Phi_0 = D_{j_0}^{1/2}$ and $\Phi_\ast = D_{j^*}^{1/2}$, the inequality (2.79) can be rewritten as

$$\frac{1}{\Phi_\ast} \lambda_{j^*+1} j^* (\Phi_0^l - \Phi_\ast^l) \geq \frac{1}{\Phi_0} \lambda_{j^*+l} (\Phi_0^l - \Phi_\ast^l).$$

Noticing

$$\lambda_{j^*+l} \left( \frac{D_{j^*+l}}{D_{j_0}} \right) \leq 0,$$

we only need to show

$$\frac{\lambda_{j^*+1} j^* \Phi_0^l (\Phi_0^l - \Phi_\ast^l)}{\lambda_{j^*+l} \Phi_\ast^l (\Phi_0^l - \Phi_\ast^l)} > 1. \quad (2.81)$$

Since $\lambda_{j^*+1} \geq \lambda_{j^*+l}$, the left hand side of (2.81) becomes

$$\frac{\lambda_{j^*+1} j^* \Phi_0^l (\Phi_0^l - \Phi_\ast^l)}{\lambda_{j^*+l} \Phi_\ast^l (\Phi_0^l - \Phi_\ast^l)} > \frac{j^* \Phi_0^l (\Phi_0^l - \Phi_\ast^l)}{l \Phi_\ast^l (\Phi_0^l - \Phi_\ast^l)}.$$

Making use of the formula (2.68) again, we obtain

$$\frac{\Phi_0^l (\Phi_0^l - \Phi_\ast^l)}{\Phi_\ast^l (\Phi_0^l - \Phi_\ast^l)} = \frac{j^* \Phi_0^l (\Phi_0^l - \Phi_\ast^l) \sum_{i=1}^l \Phi_0^{l-i} \Phi_i^{l-1}}{l \Phi_\ast^l (\Phi_0^l - \Phi_\ast^l) \sum_{i=1}^l \Phi_\ast^{l-i} \Phi_i^{l-1}} = \frac{j^* \Phi_0^l \sum_{i=1}^l \Phi_0^{l-i} \Phi_i^{l-1}}{l \Phi_\ast^l \sum_{i=1}^l \Phi_\ast^{l-i} \Phi_i^{l-1}}.$$

Observe the numerator and the denominator both have $j^* \times l$ elements represented as $\Phi_0^l \Phi_\ast^l$. But we know $\Phi_0 > \Phi_\ast$ since $D_{j_0} > D_{j^*}$, hence from (2.82), we obtain

$$\frac{\sum_{k=1}^{j^*} \sum_{i=1}^l \Phi_0^{j^*+l-i} \Phi_i^{l-1}}{\sum_{k=1}^l \sum_{i=1}^{j^*} \Phi_0^{j^*+l-i} \Phi_i^{l-1}} > \frac{\sum_{k=1}^{j^*} \Phi_0^{j^*} \Phi_i^{l-1}}{\sum_{k=1}^l \sum_{i=1}^{j^*} \Phi_0^{j^*} \Phi_i^{l-1}} = \frac{j^* \Phi_0^l \Phi_i^{l-1}}{l \Phi_\ast^l \Phi_i^{l-1}} = \frac{\Phi_0}{\Phi_\ast} > 1.$$
2.6. PROOF OF THEOREM 2.10

So

\[ \frac{j^* \Phi_0^* (\Phi_0^l - \Phi_4^l)}{l \Phi_4^l (\Phi_0^l - \Phi_4^*)} \geq 1. \]

Hence

\[ F_1 = P_1 + P_2 > 0 \] \hspace{1cm} (2.83)

is proved for \( j_0 = j^* + k \) with all \( k \in \mathbb{N} \).

**To prove** \( F_2 \geq 0 \).

Using Lemma 2.15 again, we get

\[ F_2 \geq 0. \] \hspace{1cm} (2.84)

Combining (2.83) and (2.84), we get

\[ B^2((a_s)_s, n, j_0) - B^2((a_s^*)_s, n, j^*) \geq 0 \ \forall j_0 > j^*. \] \hspace{1cm} (2.85)

Combining (2.70) and (2.85), (2.58) is proved true.

---

**Step four**

We supposed that \((a^*_s)_s \in A_{j^*}\) in the above proof. Now let us show it. First, for \( j > j^* \), from (2.26),

\[
\left( \frac{a_1^* \ldots a_j^*}{n} \right)^{\frac{1}{j}} = \left( \frac{a_1^* \ldots a_{j^*}}{n} \right)^{\frac{1}{j}} \left( a_{j^*+1}^* \ldots a_j^* \right)^{\frac{1}{j}}
\]

\[
= \left( \frac{\sqrt{\lambda_1 \ldots \sqrt{\lambda_{j^*}}}}{n} \right)^{\frac{1}{j}} \left( \left( \frac{\sqrt{\lambda_1 \ldots \lambda_{j^*}}}{n} \right)^{\frac{1}{j}} \right)^{j-j^*}
\]

\[
= \left( \frac{\sqrt{\lambda_1 \ldots \lambda_{j^*}}}{n} \right)^{\frac{1}{j} + \frac{j-j^*}{j}} = \left( \frac{\sqrt{\lambda_1 \ldots \lambda_{j^*}}}{n} \right)^{\frac{1}{j}}
\]

\[
= \left( \frac{a_1^* \ldots a_{j^*}^*}{n} \right)^{\frac{1}{j}}.
\]

Second, for \( j \leq j^* \). From (2.67), we get

\[
\left( \frac{a_1^* \ldots a_j^*}{n} \right)^{\frac{1}{j}} = \left( \frac{\sqrt{\lambda_1 \ldots \sqrt{\lambda_j}}}{n^2} \right)^{\frac{1}{j}} \geq \left( \frac{\sqrt{\lambda_1 \ldots \lambda_{j-1}}}{n^2} \right)^{\frac{1}{j-1}} = \left( \frac{a_1^* \ldots a_{j-1}^*}{n} \right)^{\frac{1}{j-1}}.
\]
Thus \((a^*_s)_s \in A_j^*\).

We can also show \((\sqrt{\lambda_s/a^*_s})_s \in \ell_2\).

\[\left(\sqrt{\frac{\lambda_s}{a^*_s}}\right)_s = \sqrt{\sum_{i=1}^{\infty} \frac{\lambda_i}{a^2_i}} = \sqrt{j^* + \frac{1}{\Lambda^2} \sum_{i=j^*+1}^{\infty} \lambda_i}. \tag{2.86}\]

When \(k(x,y)\) and \(n\) are given, \((\lambda_i)_i^j\) and \(j^*\) are determined. So \(\Lambda = n^{-\frac{3}{2}} (\lambda_1 \cdots \lambda_{j^*})^{\frac{1}{j^*}}\) is a constant. By Mercer's Theorem, \((\lambda_i)_i^j \in \ell_1\) and thus \(\sum_{i=j^*+1}^{\infty} \lambda_i\) is finite. So (2.86) is finite. Hence \((\sqrt{\lambda_s/a^*_s})_s \in \ell_2\) is proved.

**Conclusion**

Following the proof above, we get

**Corollary 2.17** Suppose \(A_j\) and \(B((a_s)_s, n, j)\) are defined as in (2.23) and (2.24). Then we have

\[B((a^*_s(j^*)), n, j^*) = \inf_{j \in \mathbb{N}} \inf_{(a_s)_s \in A_j} B((a_s)_s, n, j), \tag{2.87}\]

where

\[a^*_i = \begin{cases} \sqrt{\lambda_i} & \text{when } i \leq j^* \\ \left(\frac{\lambda_1 \cdots \lambda_{j^*}}{n}\right)^{\frac{1}{j^*}} & \text{when } i > j^*, \end{cases} \tag{2.88}\]

\[j^* = \min \left\{ j : \lambda_{j+1} < \left(\frac{\lambda_1 \cdots \lambda_{j^*}}{n^2}\right)^{\frac{1}{j^*}} \right\}. \tag{2.89}\]

Theorem 2.10 is then established.

**2.7 Proof that Lemma 2.14 cannot be improved**

**Lemma 2.18** Suppose \(A_j\) and \(B((a_s)_s, n, j)\) are defined as above. Let \(j \in \mathbb{N}\) and \((a_s)_s \in A_j\). Suppose \(j^*\) and \((a^*_s)_s\) satisfy (2.43). Then

\[\inf_{(a_s)_s \in \ell_2} \sup_{j \in \mathbb{N}} B((a_s)_s, n, j) = \inf_{j \in \mathbb{N}} \inf_{(a_s)_s \in A_j} B((a_s)_s, n, j). \tag{2.90}\]
Proof Let us prove

\[
\inf_{(a_s)_s : (\sqrt{\lambda_s/a_s})_s \in \ell_2 \ j \in \mathbb{N}} \sup B((a_s)_s, n, j) \geq \inf_{j \in \mathbb{N}} \inf_{(a_s)_s \in A_j} B((a_s)_s, n, j). \tag{2.91}
\]

Choose an \((a^*_s)_s\) to realize the infimum on the left hand side; then \((a^*_s)_s \in A_{j^*}\), where \(j^*\) is the \(j\) that realizes the inner supremum. Then

\[
\inf_{(a_s)_s : (\sqrt{\lambda_s/a_s})_s \in \ell_2 \ j \in \mathbb{N}} \sup B((a_s)_s, n, j) = \sup_{j \in \mathbb{N}} B((a^*_s)_s, n, j)
= B((a^*_s)_s, n, j^*) \geq \inf_{(a_s)_s \in A_{j^*}} B((a_s)_s, n, j^*)
\geq \inf_{j \in \mathbb{N}} \inf_{(a_s)_s \in A_j} B((a_s)_s, n, j).
\]

We have already proved

\[
\inf_{(a_s)_s : (\sqrt{\lambda_s/a_s})_s \in \ell_2 \ j \in \mathbb{N}} \sup B((a_s)_s, n, j) \leq \inf_{j \in \mathbb{N}} \inf_{(a_s)_s \in A_j} B((a_s)_s, n, j).
\]

So, equation (2.90) is proved to be true. \(\blacksquare\)
Chapter 3

Time Series Prediction Using Support Vector Machines

For sophisticated time series prediction problems, the traditional linear method often fails. It has been shown [17] that the neural network with a single hidden layer can approximate any continuous function. But a problem exists for time series prediction with neural networks: overfitting the data so that the “forecast” would have worked well at all time points in the past but fails miserably in the future. As described in Chapter 1, support vector algorithms can efficiently avoid such “overfitting” problem.

In this chapter, we will briefly introduce the time series prediction problem first. Then we will summarize the classical support vector classification and regression algorithms systematically for different cost functions. Based on this background knowledge, we will use support vector machines for time series prediction, and examine the effect of different cost functions and different kernels in the experiments. We will also apply the algorithms to the complex valued time series prediction problems. The experimental results show the good performance of support vector machines for some nonlinear time series prediction problems. The chapter concludes with a comparison with other time series prediction algorithms.
CHAPTER 3. TIME SERIES PREDICTION

3.1 Introduction

A time series is a collection of observations made sequentially in time. Given an observed time series, one may want to predict the future values of the series. This is an important task in the analysis of economic and industrial time series. For example, we might be given the closing prices for a given stock over the last month or year, and be asked to predict the closing price tomorrow. Or we might be given the weather record over the last few months, and want to know what the weather will be like the next few days. Behind these especial problems, we can give a general model of time series prediction.

Given a one-variable time series represented by \( N \) values \( x(t - N), x(t - N + 1), \ldots, x(t - 1) \), the prediction problem is to find the future value with delay \( \tau \):

\[
\hat{x}(t + \tau) := f(x(t - N), x(t - N + 1), \ldots, x(t - 1)),
\]

where \( \hat{x} \) denotes the predicted value and \( x \) is the true value. Here we call \( N \) the embedding dimension and \( \tau \) the delay.

For linear time series prediction, the “auto regression moving average” (ARMA) model and its family, such as the AR model and the MA model, play a key role in modeling for time series prediction. The basic idea of ARMA is that the value of the variable we are trying to predict is a weighted average of the values at a number of previous time points plus a weighted average of the errors of the prediction at each of those previous time points. This average will not be perfectly accurate, but the error can be used in future weighted averages and this will produce more accurate estimation. How do we determine the precise weightings to use on the averages? There are many standard statistical regression techniques. Here we give a simple example.

Suppose we are given a time series \( x(1), x(2), \ldots, x(l) \), and the prediction model is a AR model

\[
x(t + \tau) = \alpha_1 x(t - N) + \alpha_2 x(t - N + 1) + \ldots + \alpha_N x(t - 1) + \beta.
\]

In order to express it in matrix notation, we set \( x_t = (x(t - N), \ldots, x(t - 1), 1) \), the
output $y_t = x(t + \tau)$ and the weights $v = (\alpha_1, \ldots, \alpha_N, \beta)^T$, so that the AR model can be rewritten as $x_t v = y_t$. We can get $m := l - N + 1$ such equations for $t = 1, \ldots, m$ and write them together as $A v = y$, where $A = (x_1, \ldots, x_m)^T$ and $y = (y_1, \ldots, y_m)^T$.

Now we know $A$ and $y$ and want to get $v$. One of the simplest and traditional methods is to choose $v$ to minimize the sum of the least square error of the samples, $\|A v - y\|_T^2$. When $A$ has maximal rank $N + 1$, the matrix $A^T A$ is invertible, and the solution of the least square problem can be written as

$$v = (A^T A)^{-1} A^T y.$$  

For more sophisticated problems, however, the AR model often fails. It has been shown [17] that the neural network with a single hidden layer can approximate any continuous function. But a problem with neural networks is that the “forecast” would have worked well at all time points in the past but fails miserably in the future. Müller et al. applied support vector regression (SVR) in time series prediction and achieve results better than those achieved using neural networks [31].

In this chapter, we will briefly introduce the support vector classification (SVC) and SVR algorithms, and give the dual optimization problems for different cost functions. Then we will use SVC and SVR for two experiments. The first experiment considers a comparison of auto regression (AR) model and SVR for Mackey Glass data with high noise. SVR shows excellent performance. For example, the test error of SVR (using RBF kernel for 17-step ahead prediction) is only 32% of the test error of AR model. The second experiment considers a time series prediction and classification problem, where the radio data set is from the complex space $\mathbb{C}$. The prediction results show good performance by using a nonlinear kernel.

3.2 Support vector algorithms

In this section, we will briefly introduce the mathematical techniques which help us to find optimal hyperplanes, and give the dual optimization problems of support vector classification and regression (see [16, 40] for more details).
3.2.1 Basic idea of primal optimization problem

Recall the optimization problems for maximal margin classification and regression have a similar form, which is called the *primal optimization problem*:

\[
\begin{align*}
\text{minimize} & \quad f(w), \quad w \in \Gamma, \\
\text{subject to} & \quad g_i(w) < 0, \quad i = 1, \ldots, k, \\
& \quad h_i(w) = 0, \quad i = 1, \ldots, m,
\end{align*}
\]  \tag{3.1}

where the domain \( \Gamma \subseteq \mathbb{R}^n \). Here \( f(w) \) is called the *objective function*, and the remaining relations are the *inequality* and *equality* constraints.

When there are different assumptions on the objective function and the constraints, one can create different optimization problems. For the optimization problem of support vector machines, the constraints are linear, and the objective function is convex and quadratic and \( \Gamma = \mathbb{R}^n \). So it is a convex quadratic programming problem. The key idea of solving this problem is to construct a *Lagrangian function* based on the Lagrangian theory, which describes a way to characterize the solution of an optimization problem. The Lagrangian function incorporates both the information from the objective function and the constraints. The *generalized Lagrangian function* is defined as

\[
L(w, \alpha, \eta) := f(w) + \sum_{i=1}^{k} \alpha_i g_i(w) + \sum_{i=1}^{m} \eta_i h_i(w),
\]  \tag{3.2}

where the coefficients \( \alpha_i \) and \( \eta_i \) are called the *Lagrange multipliers*. To simplify the notation, we write \( \alpha \) for \( (\alpha_1, \ldots, \alpha_m) \). The expressions \( w, \eta \) have the similar meaning.

The optimization problem of support vector machines can be solved in an easier way via its dual formulation by constructing the corresponding Lagrange function. We will consider the cases of classification and regression separately.

3.2.2 Support vector classification

The primal optimization problem for maximal margin classification is

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} ||w||^2 + \frac{C}{2} \sum_{i=1}^{m} \xi_i^2, \\
\text{subject to} & \quad y_i((w, x_i) + b) \geq 1 - \xi_i, \quad \text{for } i = 1, \ldots, m.
\end{align*}
\]  \tag{3.3}
3.2. SUPPORT VECTOR ALGORITHMS

Here $\xi_i, i = 1, \ldots, m$ are the positive slack variables, which allow the margin constraints to be violated. So when the data are noisy, the overfitting problem can be avoided efficiently. Comparing with (1.16), the margin here is "soft", so $\sum_{i=1}^m \xi_i^2$ is also called the $\ell_2$-norm soft margin cost function. One can construct a similar optimization problem based on other cost functions [16, 40]. Notice that in (3.3) we do not require $\xi_i \geq 0$ because if $\xi_i < 0$, the first constraint will still hold and the optimal solution for (3.3) will coincide with the problem with the positivity constraint on $\xi_i$. Since we will use the $\ell_2$-norm soft margin cost function for our classification experiments, we give the dual optimization formulation of (3.3) below:

\[
\begin{align*}
\text{maximize} \quad & \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j \left( k(x_i, x_j) + \frac{1}{C} \delta_{ij} \right), \\
\text{subject to} \quad & \sum_{i=1}^m y_i \alpha_i = 1, \\
& \alpha_i \geq 0, \text{ for } i = 1, \ldots, m, \\
\end{align*}
\]

(3.4)

where $\delta_{ij}$ is the Kronecker $\delta$ defined to be 1 if $i = j$ and 0 otherwise. Suppose $\alpha^*$ is the solution for (3.4), the decision function is then given by

\[
\begin{align*}
f(x) := \text{sign} (g(x)) &= \text{sign} \left( \sum_{i=1}^m y_i \alpha_i^* (x_i, x) + b \right), \\
\end{align*}
\]

where $b$ is chosen so that $y_i f(x_i) = 1 - \alpha_i^*/C$ for any $i$ with $\alpha_i^* \neq 0$. 

Figure 3.1: The $\varepsilon$-insensitive band for one dimensional regression problem. Left: linear case. Right: nonlinear case.
3.2.3 Support vector regression

In the previous description, the support vector algorithm only applies to the case of classification. In fact, the idea of the support vector method, such as mapping training data nonlinearly into a higher dimensional kernel-introduced feature space and computing the hyperplane, is also a powerful technique for the case of regression [51, 16, 40].

For regression case, a so called “ε-insensitive” support vector algorithm [51] is commonly used. For the “ε-insensitive” support vector algorithm, a cost function that ignores errors within a certain distance of the true value is defined, aiming to find the hypothesis $f(x)$ with good generalization performance. Figure 3.1 shows examples of linear and nonlinear case where the one dimensional regression functions $f(x)$ are in a ε-tube. The variables $\xi$ measure the cost of the errors on the training points. These are zero for all the points inside the ε-tube.

The (linear) ε-insensitive cost function is defined by

$$c(y, f(x)) = \begin{cases} 0 & \text{if } |y - f(x)| \leq \epsilon, \\ |y - f(x)| - \epsilon & \text{otherwise}. \end{cases}$$

Similarly, one can define the quadratic ε-insensitive cost function as

$$c(y, f(x)) = |y - f(x)|^2.$$  \hfill (3.5)

Apart from these standard cost functions, there are many other choices which can be used in support vector algorithms [40], such as the Polynomial cost function: $c(y, f(x)) = \frac{1}{p} |y - f(x)|^p$, the Laplacian cost function: $c(y, f(x)) = |y - f(x)|$, and Huber’s cost function:

$$c(y, f(x)) = \begin{cases} \frac{1}{2\mu} (y - f(x))^2 & \text{if } |y - f(x)| \leq \mu, \\ |y - f(x)| - \frac{\mu}{2} & \text{otherwise}. \end{cases}$$

Figure 3.2 shows the form of the cost functions (3.5), (3.6) and (3.7) for zero and non-zero ε or µ as a function of $y - f(x)$. We will give the corresponding dual problems of (3.6) and (3.7) briefly below (see [40] for details).
3.2. SUPPORT VECTOR ALGORITHMS

Figure 3.2: The cost functions for zero and non-zero $\epsilon$ or $\mu$. Left: linear $\epsilon$-insensitive. Middle: quadratic $\epsilon$-insensitive. Right: Huber’s.

**Quadratic $\epsilon$-insensitive cost**

The dual problem which corresponds to the quadratic $\epsilon$-insensitive cost function (3.6) is

\[
\begin{align*}
\text{minimize} \quad & \frac{1}{2} \sum_{i,j=1}^{m} (\alpha_i - \tilde{\alpha}_i)(\alpha_j - \tilde{\alpha}_j)(x_i, x_j) + \frac{1}{C} \delta_{ij} \\
& - \sum_{i=1}^{m} y_i (\alpha_i - \tilde{\alpha}_i) + \epsilon \sum_{i=1}^{m} (\alpha_i + \tilde{\alpha}_i), \\
\text{subject to} \quad & \sum_{i=1}^{m} (\alpha_i - \tilde{\alpha}_i) = 0, \\
& \alpha_i, \tilde{\alpha}_i \in [0, C] \text{ for } i = 1, \ldots, m.
\end{align*}
\]

(3.8)

Suppose $\alpha$ and $\tilde{\alpha}$ are the solution of the dual problem. The regression hypothesis is

\[
f(x) = \sum_{i=1}^{m} (\alpha_i - \tilde{\alpha}_i)(x_i, x) + b,
\]

(3.9)

where $b$ is chosen so that $f(x_i) - y_i = -\epsilon - (\alpha_i - \tilde{\alpha}_i)/C$ for any $i$ with $0 < (\alpha_i - \tilde{\alpha}_i)$.

**Huber’s cost**

For Huber’s cost function, the dual problem is

\[
\begin{align*}
\text{minimize} \quad & \frac{1}{2} \sum_{i,j=1}^{m} (\alpha_i - \tilde{\alpha}_i)(\alpha_j - \tilde{\alpha}_j)(x_i, x_j) \\
& + \frac{\mu}{2C} \sum_{i=1}^{m} (\alpha_i^2 + \tilde{\alpha}_i^2) - \sum_{i=1}^{m} y_i (\alpha_i - \tilde{\alpha}_i) \\
\text{subject to} \quad & \sum_{i=1}^{m} (\alpha_i - \tilde{\alpha}_i) = 0, \\
& 0 \leq \alpha_i, \tilde{\alpha}_i \leq C.
\end{align*}
\]

The corresponding threshold $b$ of the regression hypothesis (3.9) is chosen so that $f(x_i) - y_i = -(\alpha_i - \tilde{\alpha}_i)/C$ for any $i$ with $0 \leq \alpha_i, \tilde{\alpha}_i < 1/C$.

To make all these algorithms nonlinear, we only need to use $k(x_i, x_j)$ instead of
CHAPTER 3. TIME SERIES PREDICTION

(\mathbf{x}_i, \mathbf{x}_j) in all the calculations, and the corresponding regression hypothesis \( f(\mathbf{x}) \) will become the hyperplane in the feature space defined by the kernel.

3.3 Experiments

As explained in Section 3.1, ARMA is the main model used for linear time series prediction. But the nonlinear case is much more powerful. It is difficult to solve the overfitting problem for the noisy case. As described in Sections 1.3 and 3.2, for support vector machines, slightly perturbing a training example with large margin will hardly change the hypothesis. Thus the overfitting problem can be controlled. Another feature of support vector machines is that the optimization problems are convex and hence have no local minimum. Such advantages distinguish support vector machines from other nonlinear algorithms, such as neural networks. We will demonstrate how the support vector algorithms can be employed to predict future values of a time series efficiently in this section.

We will apply support vector classification and regression for two time series prediction data sets. The first one is the Mackey Glass data, and the second one is a radio data set. The performance obtained in the example will be compared with the results of the AR model. All the experiments were performed using Matlab code.

In the experiments, we chose three different kernels. The first one is the simplest case, the input space kernel:

\[ k(\mathbf{x}_i, \mathbf{x}_j) := \langle \mathbf{x}_i, \mathbf{x}_j \rangle. \]

The other two are the RBF kernel

\[ k(\mathbf{x}_i, \mathbf{x}_j) := \exp \left( -\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2} \right), \]

and the polynomial kernel

\[ k(\mathbf{x}_i, \mathbf{x}_j) := (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + b)^p. \]

The parameters were fixed as \( \sigma^2 = 0.75 \), \( b = 1 \), \( p = 2 \), and \( C = 10^6 \) unless otherwise specified.
3.3. EXPERIMENTS

3.3.1 The Mackey-Glass equation

In this section, we will consider the time series prediction by the support vector regression on a data set generated by the Mackey-Glass differential equation. This data set was at first proposed as a model for the production of white blood cells [29]. Since then, the prediction of future values of this time series have been considered by a number of researchers [26, 31].

The Mackey-Glass equation is of the form:

\[ y'(t) = \frac{0.2y(t - \text{delay})}{1 + y(t - \text{delay})^{10}} - 0.1y(t). \]

To find the solution of this equation, we applied the MATLAB DDE23 solver\(^1\). The delay was set to be 17, the time step used was 0.1, initial condition \( x(0) = 1.2 \), and \( 0 \leq t \leq 2000 \). (We assumed \( x(t) = 0 \) for \( t < 0 \)). From the Mackey-Glass time series

\(^1\)Available at www.runet.edu/~thompson/webdtes/
We extracted 2000 input-output data pairs of the following format

\[ \{ x(t-N), x(t-N+1), \ldots, x(t-1); x(t+\tau) \}. \]

We added noise to the first 800 pairs (training data set), which were used to train the SVM. The remaining 1200 pairs (test data set) were noiseless to measure the true estimate error of the regression hypothesis. The noise added to the training data was independent Gaussian noise with signal to noise ratio \( \text{SNR} = -20 \log(0.15) \). We fixed the embedding dimension \( N = 14 \) and chose \( 1 \leq \tau \leq 25 \).

We performed support vector regression using two different cost functions: the quadratic \( \varepsilon \)-insensitive cost function and Huber's cost function. The test error \( R_{\text{test}}[f, S^m] \) of the expected error \( R(f, P) \) was obtained via the test data set based on the corresponding cost function used in the training. The test error results are shown in Tables 3.1, 3.2 and 3.3 for different noisy training set. As a comparison, we also performed prediction by using the AR model through the standard least square (LS) cost function. The results for AR models are also included in Tables 3.1, 3.2 and 3.3.
3.3. EXPERIMENTS

Figure 3.5: Test error for $\tau = 17$ step ahead prediction from $t = 1001$ to 1600. Top: AR model through least square estimate. Bottom: support vector regression. (Data set: Mackey-Glass. Kernel: polynomial. Cost function: quadratic $\varepsilon$-insensitive. Noise: Gaussian.)

For the noiseless case (Table 3.1), AR and SVR achieve similar results when we used the input space kernel as one would expect. But the hyperplane in the nonlinear kernel defined feature space outperforms the linear AR model for the 17-step and 25-step ahead prediction. The RBF kernel performs extremely well in this case.

For the noisy case (Table 3.2 and 3.3), AR performance was a little better than SVR for the 1-step prediction, but much worse for 17-step predictions of the hyperplane in the nonlinear kernel defined feature space. This can also be seen clearly from Figure 3.3, where we used the polynomial kernel, the quadratic $\varepsilon$-insensitive cost function for SVR, and the training data set with Gaussian noise. It shows that the maximal difference between the test error of SVR and the test error of AR appears at delay $\tau = 17$. Notice that the delay was set to be 17 in the Mackey-Glass equation; SVR has captured some essential components of the real nonlinear time series model.

For the 17-step prediction, Figure 3.4 shows that the desired and predicted values of the test data ($t = 1001$ to 1600) are obviously different for AR model, but almost

<table>
<thead>
<tr>
<th>Cost function</th>
<th>Quadratic cost</th>
<th>Huber's cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 step</td>
<td>17 step</td>
</tr>
<tr>
<td>S VR</td>
<td>Input space</td>
<td>3.9 x 10^-7</td>
</tr>
<tr>
<td></td>
<td>RBF</td>
<td>3.7 x 10^-7</td>
</tr>
<tr>
<td></td>
<td>Polynomial</td>
<td>3.4 x 10^-7</td>
</tr>
<tr>
<td>AR</td>
<td>—</td>
<td>3.7 x 10^-7</td>
</tr>
</tbody>
</table>

### Table 3.2: Test error on Mackey-Glass time series for training set with 15% Gaussian noise. Three different delay: 1-step, 17-step and 25-step ahead prediction.

<table>
<thead>
<tr>
<th>Cost function</th>
<th>Quadratic cost</th>
<th>Huber's cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 step</td>
<td>17 step</td>
</tr>
<tr>
<td>S VR</td>
<td>Input space</td>
<td>0.0036</td>
</tr>
<tr>
<td></td>
<td>RBF</td>
<td>0.0045</td>
</tr>
<tr>
<td></td>
<td>Polynomial</td>
<td>0.0047</td>
</tr>
<tr>
<td>AR</td>
<td>—</td>
<td>0.0021</td>
</tr>
</tbody>
</table>

We also observed that the test error curves are quite similar for different cost functions when the kernel is fixed (Table 3.2 and 3.3). Figure 3.6 shows the test error versus the delay \( \tau \) of the quadratic \( \epsilon \)-insensitive cost function and Huber's cost function for same polynomial kernel with degree \( p = 2 \). When the \( \epsilon = 0.05 \) and \( \mu = 1 \), these two algorithms have similar performance. It also shows that for a nonlinear time series, the maximal margin hyperplane in different kernel introduced feature spaces performances quite different (Table 3.2 and 3.3). With suitable choices of nonlinear kernels and parameters, the hyperplane can perform much better than the hyperplane in the input space, as well as other linear models, such as the AR model (see the test error of the input space kernel and AR in Tables 3.1, 3.2 and 3.3).

<table>
<thead>
<tr>
<th>Cost function</th>
<th>Quadratic cost</th>
<th>Huber's cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 step</td>
<td>17 step</td>
</tr>
<tr>
<td>SVR Input space</td>
<td>0.0046</td>
<td>0.0166</td>
</tr>
<tr>
<td>SVR RBF</td>
<td>0.0053</td>
<td>0.0053</td>
</tr>
<tr>
<td>SVR Polynomial</td>
<td>0.0056</td>
<td>0.0062</td>
</tr>
<tr>
<td>AR</td>
<td>—</td>
<td>0.0036</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.0212</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

However, there are some disadvantages of SVR. Firstly, the identification of the AR model took only a few seconds, while the SVR simulation took much longer. So it is wise to choose AR for linear prediction. Secondly, finding suitable kernels and suitable value of the parameters of the kernels is crucial. Different kernels and parameters can cause quite different performance. For example, in Figure 3.6, if the margin $\epsilon = 0.46$, the SVR is inferior to the AR model. The choice of suitable values in our experiments is mainly based on experience and experiments.

### 3.3.2 Radio signal

In this section, we will consider the time series prediction and classification using support vector classification on a complex data set from the Industrial Research Ltd, Lower Hutt, New Zealand\(^2\). This data set was achieved from the following experiment. In a closed room with area about 20 square meters, an antenna was fixed on the wall which generated a radio signal with fixed frequency. A receiver moving in the same room following a fixed route recorded the complex channel gain regularly. Thus a complex time series signal $x(t)$ was obtained, and the modulus of the signal changes over time. Figure 3.7 (a) displays part of the complex signal (20 wavelengths), where the point $(0, 0)$ is the origin.

If the signal is too weak to be received by a normal receiver (e.g., mobile phone)

---

\(^2\)Thanks to Paul Teal for providing this data.
CHAPTER 3. TIME SERIES PREDICTION

Figure 3.6: Test error versus delay for two different cost functions with the same kernel. (Data set: Mackey-Glass. Kernel: polynomial. Cost function: quadratic \( \varepsilon \)-insensitive and Huber's. Noise: normal.)

When the modulus is lower than some threshold, it will be very useful if an alarm can be given a short time before the signal is lower than the threshold. Then the receiver or transmitter can adjust the frequency or power used to improve the received signal quality. Our aim in this experiment is to predict whether the signal will be lower than the threshold or not before it happens by using the support vector algorithms.

Figure 3.7 (b) displays the modulus of the time series \( x(t) \) and the dashed lines are the thresholds \( A \). For \( A_1 = 0.001 \), the positive and the negative points are equal; and for \( A_2 = 0.00049 \), the positive and negative points account for 80% and 20% of the total series.

From the time series \( x(t) \in \mathbb{C} \), we extracted the input-output data pairs of four formats, see Table 3.4. There are two formats of the input \( x_t \): the complex data series \( x_t \in \mathbb{C} \) or the real data series \( x_t \in \mathbb{R} \), and two formats of the output \( y_t \): the real \( (y_t \in \mathbb{R}) \) or binary \( (y_t \in \{-1, 1\}) \) data series. For the prediction, we need to use corresponding algorithms based on the training sample's format. That is, we used SVR when \( y_t \in \mathbb{C} \) or \( y_t \in \mathbb{R} \), and used SVC when \( y_t \in \{-1, 1\} \). We call the hypothesis
3.3. EXPERIMENTS

(a) the complex time series signal

Figure 3.7: The complex time series signal. (a): the complex signal (20 wavelengths), (b): the modulus of the time series signal from t = 1 to 1000 (solid line), and the threshold (dashed line): \( \Lambda_1 = 0.001, \Lambda_2 = 0.00049 \).

<table>
<thead>
<tr>
<th>Training data pair</th>
<th>Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_t ) ( x_{(t-N)}, \ldots, x_{(t-1)} ), ( x_t \in \mathbb{C}^N )</td>
<td>( y_t ) (</td>
</tr>
<tr>
<td>( x(t-N), \ldots, x(t-1), x_t \in \mathbb{C}^N )</td>
<td>( \text{sgn}(</td>
</tr>
<tr>
<td>( (</td>
<td>x(t-N)</td>
</tr>
<tr>
<td>( (</td>
<td>x(t-N)</td>
</tr>
</tbody>
</table>

Table 3.4: Four formats of the input-output signal pairs and hypothesis of corresponding algorithms based on the radio data set.

from the algorithms as \( g(x) \), which is not a classification function for regression cases. Thus we need to convert \( g(x) \) to the final pattern recognition hypothesis \( f(x) \) as shown in Table 3.4. For each case in Table 3.4, we again used the input space kernel, the RBF kernel (with width \( 0.001 \leq \sigma \leq 1 \)) and the polynomial kernel. In order to be comparable, we chose \( \ell_2 \)-norm soft margin cost function for SVC, and quadratic \( \epsilon \)-insensitive cost function for SVR. We fixed the embedding dimension \( N = 15 \) and adjusted the delay \( 1 \leq \tau \leq 25 \). The training sample size is 600 and the test data size is 400. Once the hypothesis \( f(x) \) was found by the procedure, a prediction error \( R_{\text{test}}[f, S^m] \) of the expected error \( R(f, P) \) was obtained via the test data set by

\[
R_{\text{test}}[f, S^m] = \frac{1}{m} \sum_{i=1}^{m} \mathbb{I}\{y_i f(x_i) < 0\}.
\]

For the complex SVC and complex SVR, we did the SVC or SVR on the real
CHAPTER 3. TIME SERIES PREDICTION

Figure 3.8: Test error versus delay \( \tau \) for complex support vector classification. The estimation based on the polynomial kernel is the best for this time series.

and imaginary part of the training sample separately, achieved the corresponding coefficients as \( \alpha, b_R, \) and \( \beta, b_I \). The weight vector is the sum of two parts: \( w_R = \sum_{j=1}^{m} \alpha_j (x_{Rj} + ix_{Ij}) \) and \( w_I = \sum_{j=1}^{m} \beta_j (x_{Rj} + ix_{Ij}) \). The final hypothesis is then given as:

\[
g(x) = |(w,x) + b| = |(w_R,x) + b_R + (w_I,x) + b_I|
\]

\[
= |(w_R,x_R) + i(w_R,x_I) + b_R + (w_I,x_R) + i(w_I,x_I) + b_I|,
\]

where \( x_R, x_I \) are the real part and the imaginary part of \( x, i = \sqrt{-1} \).

The test error versus the delay \( \tau \) of complex SVC is shown in Figure 3.8. The estimate result using the polynomial kernel (solid line) achieved the lowest test error, the result using the input space kernel is in the middle, and the result using the RBF kernel turns to be the worst (nearly 50% test error). The solid line with square is the test error of AR model, which is 39.1%, while the test error of polynomial kernel is 25.6% when the delay \( \tau = 25 \).

In the next step, we compare four different conditions of the input-output sample (Table 3.4) of the polynomial kernel (see Figure 3.9). Although the test error are similar when the delay \( \tau \) is small (\( \tau \leq 5 \)), their difference is obvious as \( \tau \) increases. We can conclude two points from this figure. Firstly, the prediction using complex
3.3. EXPERIMENTS

Figure 3.9: Comparison of the test error for CSVC, CSVR, SVC and SVR. The test error of CSVC is better than that of CSVR. The test error of SVC is better than that of SVR.

input is better than that using the real input, which of course make sense since one throws away information when going from complex data to real data. For case one and case two ($x_t \in \mathbb{C}^N$), the test error is lower than that of case three and four ($x_t \in \mathbb{R}^N$). At some point, this difference is very big (see Figure 3.9). Secondly, the complex SVC’s estimation is better than the complex SVR’s estimation, and SVC’s estimation is better than the SVR’s estimation when the delay $\tau$ is large. For CSVR and SVR cases, we firstly solved a regression problem, achieved the regression hypothesis $g(x)$, then we compared $g(x)$ with the threshold $\Lambda$, and achieved the final classifier $f(x)$ (see Table 3.4). That is, we used the conditions in optimization problem of regression for a pattern recognition problem, and achieved worse results than CSVC and SVC. We conclude that the conditions in optimization problem of classification is more suitable for a pattern recognition problem.

All the results above in this experiment are based on the standard situation of equal classification problem, where the positive and negative class are both 50% of the total series (see $\Lambda_1$ in Figure 3.7). For the sampling bias condition, where the positive and negative class account for 80% and 20% of the total series (see $\Lambda_2 = 0.00049$ in Figure 3.7), we also compare all four conditions using the polynomial kernel, and get
similar results as in Figure 3.9. The complex SVC still performs the best, but the difference between each condition is not as big as in Figure 3.9.

This experiment shows that support vector type learning algorithms are useful for complex valued time series prediction as well as for real valued time series. We also saw the advantage of using a nonlinear kernel for prediction of mobile radio signal strength—a problem that to date has been largely studied standard using linear methods [45].

3.4 Discussion

We observed another interesting phenomenon in the experiment of the second data set for the comparison between SVR and AR. For the linear AR model, we chose the least square cost function. For the SVR, we chose the quadratic \( \epsilon \)-insensitive cost function where the margin \( \epsilon = 0 \). We increased \( C \) and observed the test error versus the embedding dimension \( N \) for a fixed delay \( \tau = 1 \). The experiments showed that SVR almost behaves the same as AR when \( C \leq 10^{10} \), but when \( C \geq 10^{10} \), SVR starts to behave different and the algorithm was broken when \( C \geq 10^{11} \) (see Figure 3.10).

In order to explain this phenomenon, let us analyze the SVR with the quadratic \( \epsilon \)-insensitive cost function again. In the \( \epsilon \)-insensitive cost functions, the effect of the non-zero \( \epsilon \) is to introduce an extra weight decay factor involving the dual parameters. It is shown in [16] that the solution of the optimization problem (3.8) when \( \epsilon = 0 \) is

\[
\alpha = \frac{2}{C} \left( K + \frac{1}{C} I \right)^{-1} y,
\]

where \( K \) denotes the Gram matrix \( K_{ij} = k(x_i, x_j) \) for \( i, j = 1, \ldots, m \), \( I \) the identity matrix, \( y = (y_1, \ldots, y_m)^T \). The corresponding regression function is

\[ f(x) = y^T \left( K + \frac{1}{C} I \right)^{-1} k + b, \quad (3.10) \]

where the vector \( k = (k(x_1, x), \ldots, k(x_m, x)) \).

It can also be observed from (3.8) that as \( C \to \infty \) and \( \epsilon = 0 \), the problem tends
3.4. DISCUSSION

Figure 3.10: Test error versus embedding dimension $N$ for SVR (using quadratic $\varepsilon$-insensitive cost function) and AR (using least square cost function) models. Fix delay $\tau = 1$ and margin $\varepsilon = 0$, adjust the parameter $C$. When $C \leq 10^9$, SVR and AR models have the same test error. When $C \geq 10^{10}$, SVR starts to behave different from the AR model.

to be an unconstrained least squares. The regression function (3.10) then becomes

$$f(x) = y^T K^{-1} k + b. \quad (3.11)$$

It is proved in [24] that the Gram matrix $K$ is positive semidefinite, and $K$ is non-singular iff the vectors $x_1, \ldots, x_m$ are independent. If the input vectors $x_i \in \mathcal{X}$ are from a $d$ dimension real space $\mathbb{R}^d$, and the sample size $m > d$ (it is the case in our experiments), then $x_1, \ldots, x_m$ cannot be independent. Hence $K$ is a singular matrix and $K^{-1}$ does not exist. So, when one sets $\varepsilon = 0$ and $C \to \infty$, although the theoretical analysis shows that the quadratic $\varepsilon$-insensitive cost function is consistent with the least square cost function, there has no solution to the corresponding optimization problem.

Furthermore, comparing (3.10) with (3.11), it can be seen that (3.10) makes the Gram matrix diagonal positive by adding the identity matrix when $C$ is a positive finite real number. Because $K$ is positive semidefinite, the matrix $K + \frac{1}{C} I$ is now
positive definite and invertible, so the corresponding optimization problem can be solved. Taking account of numerical errors, the experimental result is consistent with this analysis.
Chapter 4

Norm-Based Regularization of Boosting

In Chapter 1, we explained that the AdaBoost algorithm has been used with great success as a high-level learning procedure to obtain strong learners from weak learners. It has been shown that this is achieved by gradient descent on a cost function involving the margin of classification. Unfortunately, in the presence of noisy data or an overly complex function class, the algorithm tends to overfit.

In this chapter we address the problem by adding a regularization term to the overall cost function of AdaBoost. This approach provides a seamless connection between boosting and regularization networks such as support vector machines. Experimental evidence shows the feasibility of our method.

4.1 Introduction

The AdaBoost algorithm [19] can be viewed as a procedure for producing voted classifiers which minimize the sample average of an exponential cost function of the training margins. Although AdaBoost is remarkably successful in practice, AdaBoost’s cost function often places too much emphasis on examples with large negative margins. Hence it can suffer from overfitting, particularly in high noise situations.
For the problem of approximating a smooth function from sparse data, regularization techniques [47, 48] impose constraints on the approximating set of functions. Rätsch et al. [33] shows that versions of AdaBoost modified to use regularization are more robust for noisy data. Mason et al. [30] discusses the problem of approximating a smoother function based on boosting algorithms and a regularization term was also suggested to be added to the original cost function.

There is a broad range of choices for the regularization term, including many of the popular generalized additive models used in some regularization networks [22]. Since the estimates of the target function in boosting algorithms are elements of an inner product space (see next section), one natural way is to construct the regularization term as a Hilbert space constraint [30, 55].

Since kernel methods have gained popularity recently in machine learning research such as support vector machines [51], regularization networks [22], or gaussian processes [56], the regularization term can also be designed to be in a reproducing kernel Hilbert space (RKHS) [3].

In this chapter, we describe a class of algorithms (which we call NormBoost algorithms) which minimize a regularized risk functional that includes an error term and a regularization term. Here the error term may be, for instance, the original AdaBoost cost function, and the regularization term is a function either defined in the Hilbert space of [30] or a RKHS. Like AdaBoost, NormBoost performs gradient descent so as to maximally reduce the regularized risk functional at each iteration. At the same time, due to the regularization term, NormBoost approximates a smoother function than AdaBoost. That is, instead of only considering the example data, it also takes account of the capacity of the classifier. We show that NormBoost achieves better generalization performance than AdaBoost in noisy situations.

A particular advantage of the RKHS based regularization is that we can appeal to the "representer theorem" (see Theorem 4.1) which allows us to conclude that the optimal linear combination of weak hypotheses can actually be written using only \( m \) terms (where \( m \) is the number of examples). The significance of this is that it allows
us to perform gradient descent in an m-dimensional space rather than effectively in
an infinite dimensional one, as for the standard AdaBoost algorithm (for example).
One would expect that this leads to faster convergence, and indeed we demonstrate
a marked improvement in convergence rate.

4.2 Basic idea

4.2.1 The cost function

As explained in Chapter 1, the training of a learning machine commonly involves the
empirical risk functional $R_{\text{emp}}[f, S^m]$ which contains a cost function measuring the
cost incurred for the training patterns. In this chapter, we only consider the pattern
recognition problem. So we define the cost function as $c : \{-1, 1\} \times \mathbb{R} \rightarrow \mathbb{R}$. Next we
display some suitable cost functions and their derivatives which will be used in the
algorithms.

**AdaBoost Cost Function** The cost function of AdaBoost is

$$c(y_i, f(x_i)) = \exp(-y_i f(x_i)). \quad (4.1)$$

The derivative of the AdaBoost cost function with respect to $f$ is

$$c'(y_i, f(x_i)) = -y_i \exp(-y_i f(x_i)). \quad (4.2)$$

**Soft Margin Cost Function** The soft margin cost function [6], is defined as

$$c(y_i, f(x_i)) = \max(0, 1 - y_i f(x_i)) = \begin{cases} 
0 & \text{if } y_i f(x_i) \geq 1 \\
1 - y_i f(x_i) & \text{otherwise}.
\end{cases} \quad (4.3)$$

The derivative of the soft margin cost function with respect to $f$ is

$$c'(y_i, f(x_i)) = \begin{cases} 
0 & \text{if } y_i f(x_i) \geq 1 \\
-1 & \text{otherwise}.
\end{cases} \quad (4.4)$$

**Logistic Cost Function** The logistic cost function (cf. [25]) is defined as

$$c(y_i, f(x_i)) = \ln(1 + \exp(-y_i f(x_i))). \quad (4.5)$$
The derivative of the logistic cost function with respect to $f$ is

$$c'(y_i, f(x_i)) = -\frac{y_i \exp(-y_i f(x_i))}{1 + \exp(-y_i f(x_i))}. \quad (4.6)$$

**Squared Cost Function** The squared cost function is just the cost function used in (1.6),

$$c(y_i, f(x_i)) = \frac{1}{2}(y_i - f(x_i))^2. \quad (4.7)$$

The derivative of the squared cost function (4.7) with respect to $f$ is

$$c'(y_i, f(x_i)) = -(y_i - f(x_i)). \quad (4.8)$$

The treatment of each of these cost functions in the framework we present is similar.

Since our main goal will be to study modifications of $R_{\text{emp}}[f, S^m]$ to the extent of adding regularization and smoothness terms, we will use the standard cost function of AdaBoost (4.1) in the experiments. Clearly, other cost functions such as the logistic cost (4.5) are also possible [54, 57].

### 4.2.2 A Hilbert space on $X^m$

Direct minimization of $R_{\text{emp}}[f, S^m]$ may lead to bad generalization performance. Therefore we add a regularization term to the empirical risk. One possibility is to use the norm of $f$ in a Hilbert space, following [30]. These inner product space $\mathcal{H}_2^{X^m}$, with

$$\langle f, g \rangle_{\mathcal{H}_2^{X^m}} := \sum_{i=1}^{m} f(x_i)g(x_i) \quad (4.9)$$

was utilized. This is well defined for all real functions $f, g$ on $X$. Note that we only regard the inner product on the sample $X^m$ rather than on the space of all possible observations $X$.

When minimizing $R_{\text{emp}}[f, S^m]$, the Hilbert space norm

$$\|f\|^{2}_{\mathcal{H}_2^{X^m}} := \langle f, f \rangle_{\mathcal{H}_2^{X^m}} = \sum_{i=1}^{m} f^2(x_i) \quad (4.10)$$

can be used in a regularization term. This leads to our first regularized risk functional:

$$R_{\text{reg}}[f, S^m] := R_{\text{emp}}[f, S^m] + \lambda \Omega(\|f\|_{\mathcal{H}_2^{X^m}}) \text{ where } \lambda > 0. \quad (4.11)$$
Here, $\Omega$ is a monotonically increasing function and $\lambda$ is the \textit{regularization constant}. The aim is to find the function $f \in \text{span}(\mathcal{F})$ such that $R_{\text{reg}}[f]$ is minimized. The first term in (4.11) is enforcing closeness to the sample $S^m$, and the second smoothness, while $\lambda$ balances the trade-off between goodness-of-fit (small $R_{\text{emp}}$) and simplicity of the hypothesis (small $\Omega(\|f\|_{l_2}^2)$). A convenient choice for $\Omega$ is

$$
\Omega(\xi) = \frac{1}{2} \xi^2, \text{ i.e., } \Omega(\|f\|_{l_2}^2) = \frac{1}{2} \|f\|_{l_2}^2.
$$

(4.12)

### 4.2.3 Reproducing kernel Hilbert spaces

A reproducing kernel Hilbert space (RKHS) $\mathcal{H}_k$ [3] is a Hilbert space of functions defined over some bounded domain $\mathcal{X}$ with the property that, for each $x \in \mathcal{X}$, the evaluation functionals $V_x[f]$, defined as

$$
V_x[f] = f(x), \quad \forall f \in \mathcal{H}_k
$$

are linear, bounded functionals. The boundedness means that there exists a $U \in \mathbb{R}^+$ such that

$$
|V_x[f]| = |f(x)| \leq U \|f\|, \quad \forall f \in \mathcal{H}_k.
$$

It can be proved that for every RKHS, we can associate with $\mathcal{H}_k$ a positive definite function $k(x, y)$, which is called the reproducing kernel of $\mathcal{H}_k$. The kernel of $\mathcal{H}_k$ has the following \textit{reproducing property}:

$$
f(x) = \langle f(\cdot), k(\cdot, x) \rangle_{\mathcal{H}_k}, \quad \forall f \in \mathcal{H}_k, \quad \forall x \in \mathcal{X},
$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}_k}$ denotes the inner product in $\mathcal{H}_k$, i.e. $k$ is the \textit{representer of evaluation}. Using the reproducing property of the kernel $k$ we have

$$
\langle k(\cdot, x_i), k(\cdot, x_j) \rangle_{\mathcal{H}_k} = k(x_i, x_j).
$$

(4.13)

By using a RKHS, we can design our second regularized risk functional

$$
R_{\text{reg}}[f] := R_{\text{emp}}[f, S^m] + \lambda \Omega(\|f\|_{\mathcal{H}_k}) \text{ where } \lambda > 0.
$$

(4.14)
Here, $\Omega$ is a monotonic increasing function, $\| \cdot \|_{\mathcal{H}_k}$ denotes the corresponding RKHS norm,

$$\| \cdot \|_{\mathcal{H}_k} = \sqrt{\langle \cdot, \cdot \rangle_{\mathcal{H}_k}},$$

and $\lambda$ is the regularization constant as in (4.11).

### 4.2.4 Representer Theorem

It has been shown [27, 15] that for $R_{\text{emp}}$ of the form (1.5) (which measures the difference between the $f(x_i)$ and $y_i$), the minimizer of (4.14) can be written as

$$f(\cdot) = \sum_{i=1}^{m} \alpha_i k(\cdot, x_i), \quad (4.15)$$

where $k : \mathcal{X} \times \mathcal{X} \rightarrow Y$ is the kernel of the corresponding $\mathcal{H}_k$. On $S^m$ this implies $f = K\alpha$, where the matrix $K_{ij} := k(x_i, x_j)$, the weight vector $\alpha^T := (\alpha_1, \alpha_2, \ldots, \alpha_m)$, $f = (f(x_1), \ldots, f(x_m))$. This result is known as the Representer Theorem [27].

More generally, for an arbitrary cost function, the following theorem was proved in [39].

**Theorem 4.1 (Nonparametric Representer Theorem)** Suppose we are given a RKHS $\mathcal{H}$ with the positive definite real-valued kernel $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$, a training set $S^m$, a strictly monotonic increasing real-valued function $\Omega$ on $[0, \infty]$, an arbitrary cost function $c : (\mathcal{X} \times \mathbb{R})^m \times \mathbb{R}^m \rightarrow \mathbb{R}^+ \cup \{\infty\}$, a class of functions

$$T = \{ f(x) = \sum_{i=1}^{\infty} \beta_i k(\cdot, x_i), \beta_i \in \mathbb{R}, x_i \in \mathcal{X}, \|f\|_{\mathcal{H}_k} < \infty \},$$

and $\bar{T}$ the closure of $T$ in $\mathcal{H}$. Then any $f \in \bar{T}$ minimizing the regularized risk functional

$$R_{\text{reg}}[f, S^m] = R_{\text{emp}}[f, S^m] + \Omega(\|f\|_{\mathcal{H}_k}) \quad (4.16)$$

admits a representation of the form of (4.15).
This theorem shows that for a large class of algorithms minimizing a sum of an empirical risk term and a regularization term in an RKHS, the optimal solution can be written as an expansion in terms of the training examples. The optimal function $f_{opt}$ in $\mathcal{F}$ minimizing (4.14) can be written using only $m$ terms. For such an $f_{opt}$, we have

$$\|f_{opt}\|^2_{\mathcal{H}_k} = \langle f_{opt}, f_{opt} \rangle_{\mathcal{H}_k}$$

$$= \left( \sum_{i=1}^{m} \alpha_i f(\cdot), \sum_{j=1}^{m} \alpha_j f(\cdot) \right)_{\mathcal{H}_k}$$

$$= \left( \sum_{i=1}^{m} \alpha_i k(\cdot, x_i), \sum_{j=1}^{m} \alpha_j k(\cdot, x_j) \right)_{\mathcal{H}_k}$$

$$= \sum_{i,j=1}^{m} \alpha_i \alpha_j \langle k(\cdot, x_i), k(\cdot, x_j) \rangle_{\mathcal{H}_k}$$

$$= \sum_{i,j=1}^{m} \alpha_i \alpha_j k(x_i, x_j) = \alpha^T K \alpha. \quad (4.17)$$

Thus rather than dealing with all functions in the (possibly infinite dimensional) Hilbert space $\mathcal{H}$ as in traditional boosting algorithms, all one has to do for RKHS algorithms is to find a finite set of parameters $\alpha_i$ ($i = 1, \ldots, m$). Since the optimization is within a lower dimensional space, we expect that the algorithm will converge much faster than standard boosting algorithms.

### 4.3 Gradient descent

#### 4.3.1 Basic idea

This leads to the practical question of how minimizers of the regularized risk functional can be obtained efficiently. One possibility to minimize (1.5) is gradient descent. Mason et al. [30] propose such an algorithm in function space.

This involves the computation of the gradient of $R_{reg}[f, S^m]$ with respect to $f$, represented in some space, and taking a descent step in the negative gradient direction to minimize the objective function $R_{reg}[f, S^m]$. For instance the gradient could be
computed with respect to the space of the coefficients \((\alpha_1, \ldots, \alpha_T)\) or likewise with respect to \(\ell_p^m\) which was introduced via the inner product (4.9). Further choices, such as a reproducing kernel Hilbert space are possible and much of the chapter will focus on the distinctions between the different spaces and their practical performance.

We use the gradient descent algorithm (Algorithm 1) to minimize the regularized risk functional \(R_{\text{reg}}[f, S^m]\). At each step we compute the gradient with respect to \(f\) or one of its parametric representations via \(\alpha\), take a step in the negative direction of the gradient, and repeat the procedure until convergence.

**Algorithm 1** Gradient descent algorithm

Training data \(S^m\), cost function \(c\), regularizer \(\Omega(\cdot)\), inner product \(\langle \cdot, \cdot \rangle\), tolerance \(\varepsilon\), step length \(\Lambda\) (unless line-search selected)

for \(t := 0\) to \(T\) do

Compute \(g := \nabla_f R_{\text{reg}}[f, S^m]\).

if linesearch then

\[ \Lambda = \arg\min_\gamma R_{\text{reg}}[f - \gamma g, S^m]. \]

end if

\[ f \rightarrow f - \Lambda g. \]

end for \(\|\nabla_f R_{\text{reg}}[f, S^m]\| \leq \varepsilon\)

If we perform descent in the space of \(\alpha\) rather than \(f\), we have to replace the derivatives in \(f\) by the corresponding terms in \(\alpha\) and modify the update equations in \(f\) accordingly.

Since a unit step in the direction of the negative gradient of \(R_{\text{reg}}[f]\) does not necessarily guarantee that \(R_{\text{reg}}[f]\) will decrease, it is advantageous in many cases to perform a line-search in the direction of \(\nabla R_{\text{reg}}[f]\), i.e. seek the step size \(\Lambda\) such that \(R_{\text{reg}}[f - \Lambda \nabla R_{\text{reg}}[f]]\) is minimized. One can obtain the approximately optimal step size \(\Lambda\) by, for instance, starting with an initial guess of an interval, and if no minimum can be found strictly inside the interval, doubling its size. (see, for example, [40] chapter 6.)
4.3.2 A lower bound on $R_{\text{reg}}[f, S^m]$

Before we proceed with the actual implementation details we consider a suitable stopping rule. In this subsection, we will show that the size of the gradient is a good criterion to assess the quality of the current solution.

**Special case**

The following theorem gives a lower bound on $R_{\text{reg}}[f, S^m]$ for the regularization term as $\Omega(||f||) = \frac{1}{2}||f||^2$.

**Theorem 4.2 (Lower Bound on Primal Objective Function)** Denote by $R_{\text{emp}}[f, S^m]$ a convex and differentiable functional on a Hilbert space $H$ and consider the regularized risk functional

$$R_{\text{reg}}[f, S^m] = R_{\text{emp}}[f, S^m] + \lambda \Omega(||f||) \quad \text{where } \lambda > 0 \quad (4.18)$$

Then for any $f, \Delta f \in H$, and for $\Omega(||f||) = \frac{1}{2}||f||^2$,

$$R_{\text{reg}}[f, S^m] - R_{\text{reg}}[f - \Delta f, S^m] \leq \frac{1}{2\lambda}||\nabla R_{\text{reg}}[f, S^m]||^2. \quad (4.19)$$

**Proof** We start the proof in a more general way which will be useful to give a lower bound on $R_{\text{reg}}[f, S^m]$ for the general regularization function $\Omega$ in Section 4.3.2. We assume that $f - \Delta f$ is the minimizer of $R_{\text{reg}}[f, S^m]$ since proving the inequality for the minimizer is sufficient. Since $R_{\text{emp}}$ is convex and differentiable we know that

$$R_{\text{emp}}[f, S^m] - R_{\text{emp}}[f - \Delta f, S^m] \leq \langle \Delta f, \nabla R_{\text{emp}}[f, S^m] \rangle. \quad (4.20)$$

Therefore we may bound $R_{\text{reg}}[f, S^m] - R_{\text{reg}}[f - \Delta f, S^m]$ by

$$\rho(f, \Delta f) := R_{\text{reg}}[f, S^m] - R_{\text{reg}}[f - \Delta f, S^m]$$

$$= R_{\text{emp}}[f, S^m] - R_{\text{emp}}[f - \Delta f] + \lambda \Omega(||f||) - \lambda \Omega(||f - \Delta f||)$$

$$\leq \langle \Delta f, \nabla R_{\text{emp}}[f, S^m] \rangle + \lambda (\Omega(||f||) - \Omega(||f - \Delta f||)). \quad (4.21)$$

This is in particular true for the minimization of $\rho(f, \Delta f)$ with respect to $\Delta f$. This minimum satisfies

$$\nabla_{\Delta f} \rho(f, \Delta f) = 0$$
and therefore

\[ \nabla R_{\text{emp}}[f, S^m] - \lambda \nabla \Delta f \Omega(\| f - \Delta f \|) = \nabla R_{\text{emp}}[f, S^m] + \lambda \Omega(\| f - \Delta f \|) \frac{f - \Delta f}{\| f - \Delta f \|} = 0. \]  

(4.22)

Here we used

\[ \nabla \Delta f(\| f - \Delta f \|) = \nabla \Delta f(f - \Delta f, f - \Delta f) \frac{1}{2} \]
\[ = \frac{\nabla \Delta f(f - \Delta f, f - \Delta f)}{2 (\| f - \Delta f \|)} \]
\[ = - \frac{f - \Delta f}{\| f - \Delta f \|}. \]

For the special case where \( \Omega(\| f \|) = \frac{1}{2} \| f \|^2 \), we get

\[ \Omega'(\| f - \Delta f \|) = \| f - \Delta f \|. \]

From (4.22) we have

\[ \nabla R_{\text{emp}}[f, S^m] + \lambda \| f - \Delta f \| \frac{f - \Delta f}{\| f - \Delta f \|} = 0. \]

So (4.21) is minimized by

\[ \Delta f = \frac{\nabla R_{\text{emp}}[f, S^m]}{\lambda} + f. \]

Substituting this back into the upper bound (4.21) on \( \rho(f, \Delta f) \), also noticing that when \( \Omega(\| f \|) = \frac{1}{2} \| f \|^2 \),

\[ \nabla R_{\text{reg}}[f, S^m] = \nabla R_{\text{emp}}[f, S^m] + \lambda f, \]

we have

\[ \rho(f, \Delta f) \leq \left( \frac{\nabla R_{\text{emp}}[f, S^m]}{\lambda} + f, R_{\text{emp}}[f, S^m] \right) + \frac{\lambda}{2} \left( \| f \|^2 - \left\| R_{\text{emp}}[f, S^m] \right\|^2 \right) \]
\[ = \frac{1}{2\lambda} \| R_{\text{emp}}[f, S^m] \|^2 + 2\lambda \langle f, \nabla R_{\text{emp}}[f, S^m] \rangle + \lambda^2 \| f \|^2 \]
\[ = \frac{1}{2\lambda} \| \nabla R_{\text{reg}}[f, S^m] \|^2. \]

Inequality (4.19) is proved. \[ \blacksquare \]
General cases

For general $\Omega$, matters are somewhat more delicate. We can rewrite (4.22) as

$$\lambda \Omega'(||h||) \frac{h}{||h||} = -\nabla R_{\text{emp}}[f, S^m], \text{ where } h := f - \Delta f. \quad (4.23)$$

This equation can be solved for

$$h = -\omega \frac{\nabla R_{\text{emp}}[f, S^m]}{||\nabla R_{\text{emp}}[f, S^m]||}, \text{ for some } \omega > 0. \quad (4.24)$$

This leads to

$$\lambda \Omega'(\omega) = \zeta, \text{ where } \zeta = ||\nabla R_{\text{emp}}[f, S^m]||.$$

If the function $\Omega' : \mathbb{R}^+ \to \mathbb{R}$ is invertible, the equation $\Omega'(\omega) = \zeta/\lambda$ can be solved for $\omega$ and we obtain

$$\omega = (\Omega')^{-1}(\zeta/\lambda).$$

Substituting $\omega$ back into (4.24), we achieve

$$f - \Delta f = h = - (\Omega')^{-1} \left( \frac{\zeta}{\lambda} \right) \frac{\nabla R_{\text{emp}}[f, S^m]}{||\nabla R_{\text{emp}}[f, S^m]||}.$$

This value of $f - \Delta f$ can then be used in a lower bound for $R_{\text{emp}}[f, S^m]$. At the same time, it also offers a cheap replacement for the line search strategy in the gradient descent algorithm, especially if the evaluation of $R_{\text{emp}}[f, S^m]$ is expensive in comparison to the other terms of the regularized risk functional. Finally note that these bounds are tight if the linearization of $R_{\text{emp}}[f, S^m]$ is exact. Therefore it makes an excellent stopping criterion.

### 4.3.3 Gradient descent in function space

Let us start with gradients in function space $\mathcal{F}$. Here the regularized risk functionals are given by (4.11) and (4.14). There is a wide choice for the regularization operator $\Omega$. In the following gradient descent calculation and experiments, we choose $\Omega(\xi) = \frac{1}{2} \xi^2$. Hence we have

$$R_{\text{reg}}[f] = \frac{1}{m} \sum_{i=1}^{m} c(y_i, f(x_i)) + \frac{1}{2} \xi^2, \quad (4.25)$$
We define
\[ \beta_i := \frac{1}{m} c'(y_i, f(x_i)), \quad i = 1, \ldots, m, \] (4.26)
and the vector
\[ \beta^T = (\beta_1, \beta_2, \ldots, \beta_m). \] (4.27)
Thus the vector \( \beta \) can be computed from the cost function.

\( \ell_2^X \) NormBoost

Consider the regularized risk functional (4.11) first. It can be expressed as
\[ R_{\text{reg}}[f] = \frac{1}{m} \sum_{i=1}^{m} c(y_i, f(x_i)) + \frac{\lambda}{2} \| f \|_{\ell_2^X}^2, \] (4.28)
where we still use (4.1) as the cost function. In this case, the gradient of the cost function is
\[ \nabla f_{\text{emp}}[f, S^m](x) = \beta, \] (4.29)
where \( f \) is the vector of the values of \( f \) on \( X^m \) as defined before. Equation (4.28) can be rewritten as:
\[ \nabla f_{\text{reg}}[f, S^m](x) = \beta + \lambda f, \]
and we obtain the following update rules for \( f \), given a step size \( \Lambda \).
\[ f_{t+1} = f_t - \Lambda (\beta + \lambda f). \] (4.30)
For computational reasons we represent \( f \) as a linear combination of functions in a finite dimensional space. With the expansion of \( f(x) \) as in (1.26) the update rule for the weights becomes
\[ \alpha_{t+1} = \alpha_t - \Lambda (F^{-1}\beta + \lambda \alpha_t), \] (4.31)
where the matrix \( F_{ij} := f_j(x_i) \). The inverse of \( F \) is required in (4.31). \( F \) may not be invertible, and it is also expensive to compute \( F^{-1}\beta \), hence we do not use the gradient descent in function space for \( \ell_2^X \) norm regularization.
Next we consider the regularized risk functional (4.14) with the regularizer term as in (4.25). The decision function is defined by (4.15). Combining with (1.5), the gradient of the regularized risk functional can be expressed as

\[ \nabla_f R_{\text{reg}}[f] = \nabla_f \left( \frac{1}{m} \sum_{i=1}^{m} c(y_i, f(x_i)) + \frac{\lambda}{2} \|f\|_{\mathcal{H}_k}^2 \right) \]

\[ = \sum_{i=1}^{m} \frac{1}{m} c'(y_i, f(x_i))k(\cdot, x_i) + \lambda f \]

\[ = \sum_{i=1}^{m} \beta_i k(\cdot, x_i) + \lambda f. \quad (4.32) \]

We obtain the update rules for \( f \):

\[ f_{t+1} = f_t - \Lambda \left( \sum_{i=1}^{m} \beta_i k(\cdot, x_i) + \lambda f \right) \]

\[ = \sum_{i=1}^{m} \alpha_i k(\cdot, x_i) - \Lambda \left( \sum_{i=1}^{m} \beta_i k(\cdot, x_i) + \lambda \sum_{i=1}^{m} \alpha_i k(\cdot, x_i) \right). \quad (4.33) \]

Hence the update rule for the weights becomes

\[ \alpha_{t+1} = \alpha_t - \Lambda (\beta + \lambda \alpha_t). \quad (4.34) \]

### 4.3.4 Gradient descent in coefficient space

For comparison, consider the corresponding updates for gradient descent in coefficient space. We will describe the update rules for \( \ell_2^m \) norm regularization and RKHS norm regularization separately. In order to compare easily, we use the same regularized risk functional, cost function, and function space as before.

**\( \ell_2^m \) NormBoost**

In the \( \ell_2^m \) norm regularization as above, the gradient descent of the regularization functional is

\[ \nabla_\alpha R_{\text{reg}}[f, S^m] = \nabla_\alpha \left( \frac{1}{m} \sum_{i=1}^{m} c(y_i, f(x_i)) + \frac{\lambda}{2} \|f\|_{\ell_2^m}^2 \right) \]

\[ = F^T \beta + \lambda F^T \alpha. \quad (4.35) \]
Hence the update rule for the weights is

$$\alpha_{t+1} = \alpha_t - \Lambda F^T (\beta + \lambda F \alpha_t)$$  \hspace{1cm} (4.36)

**RKHS NormBoost**

It is straightforward to see that in the case of RKHS Norm regularization as above, we obtain

$$\nabla_\alpha R_{\text{reg}}[f] = K \beta + \lambda K \alpha$$  \hspace{1cm} (4.37)

$$\alpha_{t+1} = \alpha_t - \Lambda K \beta - \Lambda \lambda K \alpha_t.$$  \hspace{1cm} (4.38)

In other words, the updates from (4.31) are multiplied by the kernel matrix $K$ to obtain the update rules in the coefficient space. This means that we are performing gradient descent in a space with respect to the metric given by $K$ rather than the Euclidean metric.

**Other regularization operators**

Beyond the function space based regularization functions, we may also use regularization directly in coefficient space. The family of $\ell_p$ norms is particularly useful here. Next we show two examples in this case.

For $\Omega(f) := \frac{1}{2} \| \alpha \|_{\ell_2}^2$, the gradient of the regularized risk functional is

$$\nabla_\alpha R_{\text{reg}}[f, S^m] = K \beta + \lambda \alpha.$$  \hspace{1cm} (4.39)

So the update rule is

$$\alpha_{t+1} = \alpha_t - \Lambda (K \beta + \lambda \alpha).$$

For $\Omega(f) := \| \alpha \|_{\ell_1}$, the gradient of the regularized risk functional is

$$\nabla_\alpha R_{\text{reg}}[f, S^m] = K \beta + \lambda \text{sign} (\alpha).$$  \hspace{1cm} (4.40)

So the update rule is

$$\alpha_{t+1} = \alpha_t - \Lambda (K \beta + \lambda \text{sign} (\alpha)).$$
4.4. EXPERIMENTS

<table>
<thead>
<tr>
<th>Regularization</th>
<th>$\frac{1}{2} | f |_H^2$</th>
<th>$| f |_C$</th>
<th>$\frac{1}{2} | f |_{\ell_2^\infty}^2$</th>
<th>$| f |_{\ell_2^m}$</th>
<th>$\frac{1}{2} | \alpha |_2^2$</th>
<th>$| \alpha |_{\ell_1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient in $f$</td>
<td>$\alpha$</td>
<td>$\frac{\alpha}{\sqrt{\alpha^T K \alpha}}$</td>
<td>$K \alpha$</td>
<td>$\frac{K \alpha}{\sqrt{\alpha^T K^T K \alpha}}$</td>
<td>$-$</td>
<td>$-$</td>
</tr>
<tr>
<td>Gradient in $\alpha$</td>
<td>$K \alpha$</td>
<td>$\frac{K \alpha}{\sqrt{\alpha^T K \alpha}}$</td>
<td>$K^T K \alpha$</td>
<td>$\frac{K^T K \alpha}{\sqrt{\alpha^T K^T K \alpha}}$</td>
<td>$\alpha$</td>
<td>\text{sign}(\alpha)</td>
</tr>
</tbody>
</table>

Table 4.1: Different regularization functionals and their gradients in function space and coefficient space.

Table 4.1 gives an overview of different regularization operators and their gradients in the RKHS inner product space.

4.4 Experiments

In order to compare the performance of NormBoost and AdaBoost, we study the performance of both algorithms on three artificial datasets. The main points we want to assess are the classification accuracy and the convergence speed. We use $R_{P[f]}$ as the expected error of a combined classifier $f$ with respect to the probability distribution $P$ on $Z = \mathcal{X} \times \{-1, 1\}$, and $R_{test[f, S^m]}$ as the estimate error of $R_{P[f]}$ which is the approximation of $R_{P[f]}$ from the training set $S^m$.

Dataset 1 The first dataset was generated in a two dimensional square $L = \{x \in \mathbb{R}^2 : \|x\|_{\ell_\infty} \leq 1\}$. The true classification function to be learned was 1 when the points were in $L_{positive} = \{(x_1, x_2) \in \mathbb{R}^2 : x_1 \geq x_2\}$, and $-1$ otherwise. For each trial a training sample $S = \{(x_i, y_i)\}_{i=1}^{50}$ of size 50 was generated with the $x_i$ drawn independently at random from a uniform distribution on $L$.

Dataset 2 The second dataset was generated by drawing $d$ dimensional points uniformly from $[-1, 1]^d$, and using them as inputs to a map defined by a mixture of $n$ Gaussian bumps with width chosen from a uniform distribution on the interval $[0.1, 1]$ and weights chosen from a uniform distribution on the interval $[0, 1]$. Patterns were split into two classes.

Dataset 3 The third dataset is Banana, which is available at
For both the first and second datasets, independent label noise of intensity 5%, 10%, 15% and 20% was applied to the labels $y_i$. Once a hypothesis $f$ was found by the procedure, an estimate error $R_{\text{test}}[f, S^m]$ of the expected error $R(f, P)$ was obtained via 10000 independently drawn test points. All of the experiments were repeated 10 times with the examples randomly selected for training and test purposes. The results were then averaged over the 10 repeats.
4.4. EXPERIMENTS

4.4.1 $\ell_2^{X^m}$ NormBoost and AdaBoost

For all the experiments in $\ell_2^{X^m}$ NormBoost, axis-orthogonal hyperplanes (also called decision stumps) were produced by the weak learner.

Figure 4.1 (first dataset) shows the test error $R_{test}[f, S^m]$ versus the regularization parameter $\lambda$ with different noise levels. The $\lambda = 0$ points show the test error of AdaBoost. With no noise, AdaBoost shows better performance than $\ell_2^{X^m}$ NormBoost. But when the training sample is noisy, the lowest test error (see Figure 4.1) is achieved at some $\lambda^* > 0$ instead of the original points. The stars in Figure 4.1 symbolize the test error of AdaBoost (with $\lambda = 0$). We notice that there is a jump at $\lambda = 0$ which will be discussed in Section 4.5.

Margin distribution graphs [37] are useful to analyze the experimental results. We know that a large positive margin can be interpreted as a “confident” correct classification. AdaBoost tends to increase the margins associated with examples and converge to a margin distribution in which most examples have large positive margins. Hence, AdaBoost can improve the generalization error of a classifier when there is no
noise. But when there is noise in the training set, AdaBoost generates an overfitting classifier by trying to classify the noisy points with positive margins. In fact, in most cases, the boosting algorithm will modify the training sample distribution to force the learner to concentrate on its errors, and will thereby force the learner to concentrate on learning noisy examples.

The NormBoost algorithm, because of the balancing influence of the regularization term, tends to find a smoother classifier and converge to a margin distribution in which some examples may have negative margins.

Figure 4.2 (first dataset) shows the margin distributions after 10000 iterations for AdaBoost, $\ell_2^X$ NormBoost with $\lambda = 0.1$, and $\ell_2^X$ NormBoost with $\lambda = 0.12$, indicated by solid, dotted, and dashed lines, respectively. With noise, about 10% of the training data are classified incorrectly, while the margins of more than 75% of the points are bigger than those of AdaBoost.

The convergence of the regularized risk functional versus the iteration time $t$ is
4.4. EXPERIMENTS

Figure 4.4: Test error versus $\lambda$ for RKHS NormBoost Algorithm. (Noise level = 20%)

shown in Figure 4.3 (first dataset), where the curve for $\lambda = 0$ depicts the $R_{reg}$ of AdaBoost and the other curves depict that of $\ell_2^{Xm}$ NormBoost with different $\lambda$ levels. Our observation is that the regularized risk functionals of $\ell_2^{Xm}$ NormBoost converge much faster than AdaBoost. Notice that the number of iterations is plotted on a log scale.

4.4.2 RKHS NormBoost and AdaBoost

For all the experiments on RKHS NormBoost, we chose Gaussian radial basis functions with width $\sigma$ as the kernel function class, i.e.

$$F_{\sigma} = \left\{ x \mapsto \exp \left( -\frac{1}{\sigma^2} ||x - x_i||^2 \right), \; i = 1, \ldots, m \right\},$$

(4.41)

and $\Omega(f(\alpha)) := \frac{1}{2} ||f(\alpha)||_{\mathcal{H}_k}^2$.

Figure 4.4 and 4.5 (second dataset) show the test error as a function of $\lambda$ for a range of different kernel widths $\sigma$ and different noise levels. The regularization
parameter $\lambda$ is plotted on a log scale. As expected, the curves have their lowest point at some $\lambda^* > 0$ for some appropriate values of $\sigma$. Hence, there are two parameters that effectively influence the generalization performance of RKHS NormBoost: $(\lambda, \sigma)$.

This observation leads us to consider the relationship between RKHS NormBoost and support vector machines. Support vector machines are linear classifiers that use the maximum margin hyperplane in a feature space also defined by a kernel function. Using the same notation $X^m$, we can denote the hypothesis class implemented by support vector machines on $X^m$ with weight vector bounded by $R_w$:

$$
\mathcal{F}_{R_w} = \left\{ x \mapsto \sum_{i=1}^{m} \alpha_i k(x, x_i) : \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k(x_i, x_j) \leq R_w^2 \right\},
$$

where $0 \leq \alpha_i \leq C$, for $i = 1, \ldots, m$ and $C \in \mathbb{R}$.

Recalling the output of RKHS NormBoost is $f(x) = \sum_{i=1}^{m} \alpha_i k(x, x_i)$ and

$$
\| f \|_{\mathcal{H}_k} = \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j k(x_i, x_j) < \infty,
$$
4.4. EXPERIMENTS

Figure 4.6: $R_{reg}$ versus the iteration time $t$ for RKHS NormBoost Algorithm. Left: Gradient in Function Space. Right: Gradient in Coefficient Space. Observe that for a noise level of 20%, $\sigma = 0.2$, $\lambda = 0.05$ gave the smallest test error (26%). For these values of $\sigma$ and $\lambda$, the algorithm converged in only two steps.

by the definition of the RKHS, we find that RKHS NormBoost has the same hypothesis class form (4.42) as support vector machines. The two parameters $(C, \sigma)$ or $(\nu, \sigma)$ (see [16]) play important roles in support vector machines. Similarly the parameters $(\lambda, \sigma)$ effectively determine the character of an RKHS NormBoost algorithm.

The choice of parameterization will have an impact on the performance of the algorithm. Fixing the noise level (20%) and $\sigma = 0.2$, for different values of $\lambda$ on our second toy problem, Figure 4.6 (second dataset) graphs the convergence speed for both experiments: one gradient in the function space and another in the weight space. Notice that $R_{reg}[f]$ converges in less than 100 iterations of RKHS NormBoost while $R_{reg}[f]$ is still decreasing after 1000 iterations for AdaBoost. We also observed that the gradient descent in function space is never slower (in terms of number of iterations) and is usually considerably faster. Furthermore it is computationally less expensive, also on a per-iteration basis.

In Figure 4.6, different curves correspond to different values of $\lambda$ from the long-dashed curve in Figure 4.4. In Figure 4.6, the long-dashed curve corresponds to
the value $\lambda = 0.05$ that achieves the minimum test error in Figure 4.4. For this "good" choice of $\lambda$, RKHS NormBoost converged in only 3 steps when $R_{\text{reg}}$ is defined using the gradient in function space and only 10 steps when $R_{\text{reg}}$ is defined using the gradient in coefficient space.

The convergence properties of $R_{\text{reg}}[f]$ are similar in the second and third datasets; see Figures 4.7, 4.8 (second dataset), 4.9 and 4.10 (third dataset). Gradient descent in the function space $\mathcal{F}$ with respect to the Euclidean metric is smoother than in the weight space $\alpha$ with respect to the metric given by the kernel matrix $K$. The performance of the RKHS NormBoost shows up more clearly in Figure 4.9 and 4.10, where the results for the dataset BANANA are given. The results show that the performance of RKHS NormBoost is comparable to other learning algorithms for suitable choice of $\lambda$ and $\sigma$. In this experiment, the lowest classification error achieved was 10.43% ± 0.47%, corresponding to $\lambda = 3$ and $\sigma = 0.6$. 

Figure 4.7: Test error versus $\lambda$. Solid curve: Gradient in Function Space. Dashed curve: Gradient in Coefficient Space.
The experiments also show that the dimension and the sample size influence the convergence performance of $R_{reg}^f$. Figure 4.11 shows that for low sample size, the convergence speed of $R_{reg}^f$ shows a big difference between the gradient in function space and in coefficient space for low dimensions, but it is hard to tell the difference for high dimension (dimension=8). On the other hand, if we increase the sample size, then the difference appears for high dimensions (see Figure 4.12).

4.5 Discussion

Both the experiments in $\ell_2^X$ NormBoost and RKHS NormBoost show a significant improvement in the convergence speed, which is one benefit of the regularization term. We described the regularization term as a smoothness function in Section 4.2, where
one function is said to be smoother than another one within a class of functions if it oscillates less. If we look at the functions in the frequency domain, a function is said to be smoother than another one if it has less energy at high frequency. The high frequency content of a function can be measured by first high-pass filtering the function, and then measuring the power. This means that we compute the norm of the function after the high-pass filtering [22]. Hence, the minimizer of the regularized risk functional (4.11) has not only less cost on the training samples, but also less high frequency content. In the frequency domain, such an optimal point in function space is easier to find than another one with more high frequency content. Hence adding a regularization term leads to much faster convergence than the standard AdaBoost algorithm.

For the RKHS NormBoost, there is another important reason for such remarkably faster convergence speed. Based on Theorem 4.1, the optimal regularized risk functional contains no more than $m$ terms, where $m$ is the sample size. This is normally
4.5. DISCUSSION

Figure 4.10: $R_{reg}$ versus the iteration time $t + 1$ for RKHS NormBoost algorithm for BANANA dataset. Gradient descent in the RKHS space with respect to the Euclidean metric is smoother than in the weight space $\alpha$ with respect to the metric given by the kernel matrix $K$. For the same $\lambda$, they converge to the same point.

much less than the number of iterations $T$ in standard boosting algorithms. Hence the search for the optimal combination need only be performed in an $m$-dimensional space instead of dealing with a possibly infinite dimensional space (see Figure 4.6).

A curious feature of the graphs in Figure 4.1 for non-zero noise levels is that there is a discontinuity at $\lambda = 0$. Although as we have argued we need to regularize when there is noise, and thus want to set $\lambda > 0$, it is an effect that cries out for an explanation. We conjecture that with no regularization the cost surface is rank degenerate and the gradient descent procedure implemented by the NormBoost algorithm essentially drifts around a subspace in directions where the gradient is zero. As soon as one sets $\lambda > 0$, this degeneracy disappears. In numerical experiments we have found that setting $\lambda$ to even extremely small values (such as $10^{-12}$) suffices to bring the test error away from the point marked by a * in Figure 4.1.
Dimension is: 4.

Dimension is: 6.

Dimension is: 8.

Figure 4.11: $R_{reg}$ versus the iteration time $t$. Change the function dimension and keep the sample size fixed (sample size = 200).

### 4.6 Generalization performance

Next we will analyze the generalization performance of two function classes in the $\ell_2^X$ space. We use $\mathcal{F}_1(\mathcal{H}, C)$ and $\mathcal{F}_2(\mathcal{H}, V)$ to denote two different function classes:

$$\mathcal{F}_1(\mathcal{H}, C) = \left\{ f := \sum_{t=1}^{T} \alpha_t f_t, \ T \in \mathbb{N}, \ f_t \in \mathcal{H}, \ \|\alpha\|_\ell^2 \leq C \right\},$$

$$\mathcal{F}_2(\mathcal{H}, V) = \left\{ f := \sum_{t=1}^{T} \alpha_t f_t, \ T \in \mathbb{N}, \ f_t \in \mathcal{H}, \ \|f\|_{\ell_\infty^X} \leq V \right\},$$

where $\mathcal{H}$ is the base classifier space. Suppose $\|f\|_{\ell_\infty^X} \leq B$ for all $f \in \mathcal{H}$, $\alpha_i \geq 0$ for all $i \in \mathbb{R}$. The following theorem shows that function classes $\mathcal{F}_1(\mathcal{H}, C)$ and $\mathcal{F}_2(\mathcal{H}, V)$ can be bounded by each other if we add a condition on the inner product $\langle f_i, f_j \rangle_{\ell_\infty^X}$.

**Theorem 4.3** Given $m$ points $x_1, \ldots, x_m \in X$, $\mathcal{F}_1(\mathcal{H}, C)$ and $\mathcal{F}_2(\mathcal{H}, V)$ are defined as above. Suppose $\langle f_i, f_j \rangle_{\ell_\infty^X} \geq \beta^2$ for all distinct $f_i, f_j$ in $\mathcal{H}$,
4.6. GENERALIZATION PERFORMANCE

Sample size is: 200.
Sample size is: 1000.
Sample size is: 2000.

Figure 4.12: $R_{reg}$ versus the iteration time $t$. Change the sample size and keep the function dimension fixed (dimension = 8).

where $\beta > 0$. Then the relationship between these two function classes is:

$$\mathcal{F}_1(\mathcal{H}, C) \subseteq \mathcal{F}_2(\mathcal{H}, BC), \quad (4.43)$$

and

$$\mathcal{F}_2(\mathcal{H}, V) \subseteq \mathcal{F}_1\left(\mathcal{H}, \frac{V}{\beta}\right), \quad (4.44)$$

Proof Part A: to prove the inequality (4.43). From the definition of the inner product, we have

$$\|f\|^2_{\mathcal{F}_2} = \langle f, f \rangle = \frac{1}{m} \sum_{i=1}^{m} f^2(x_i)$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left( \sum_{j=1}^{T} \alpha_j f_j(x_i) \sum_{k=1}^{T} \alpha_k f_k(x_i) \right)$$

$$= \sum_{j=1}^{T} \sum_{k=1}^{T} \alpha_j \alpha_k \left( \frac{1}{m} \sum_{i=1}^{m} f_j(x_i) f_k(x_i) \right)$$
\[ T = \sum_{j=1}^{T} \sum_{k=1}^{T} \alpha_j \alpha_k \langle f_j, f_k \rangle. \]  

Since \( \|h\|_{\ell_\infty} \leq B \) for all \( h \in \mathcal{H} \), we can get:

\[ \langle f_j, f_k \rangle_{\ell_2} = \frac{1}{m} \sum_{i=1}^{m} f_j(x_i) f_k(x_i) \leq \frac{1}{m} \left( mb^2 \right) = B^2. \]

Also noticing that \( \|\alpha\|_{\ell_1} \leq C \) holds for the \( f \in \mathcal{F}_1(\mathcal{H}, C) \), (4.45) can be written as

\[ \|f\|_{\ell_2}^2 \leq B^2 \left( \sum_{j=1}^{T} \sum_{k=1}^{T} \alpha_j \alpha_k \right) = B^2 \left( \sum_{j=1}^{T} \alpha_j \right)^2 \leq B^2C^2. \]

Hence, (4.43) is proved.

**Part B:** to prove the inequality (4.44). For the function \( f \in \mathcal{F}_2(\mathcal{H}, V) \), we have \( \|f\|_{\ell_2} \leq V \). Combining with equation (4.45), we get

\[ V^2 \geq \|f\|_{\ell_2}^2 = \sum_{j=1}^{T} \sum_{k=1}^{T} \alpha_j \alpha_k \langle f_j, f_k \rangle. \]

Using the condition \( \langle f_i, f_j \rangle_{\ell_\infty} \geq \beta^2 \), we have

\[ V^2 \geq \beta^2 \sum_{j=1}^{T} \sum_{k=1}^{T} \alpha_j \alpha_k = \beta^2 \left( \sum_{j=1}^{T} \alpha_j \right)^2 = \beta^2 \|\alpha\|_{\ell_1}^2. \]

Hence, \( \|\alpha\|_{\ell_1} \leq \frac{V}{\beta} \) holds, which implies (4.44).

From Theorem 4.3, we can see that if \( \beta = 1 \), i.e. \( \forall i, j, \langle f_i, f_j \rangle_{\ell_\infty} = 1, \mathcal{F}_1 = \mathcal{F}_2 \) holds. In fact, the condition \( \beta > 0 \) is a very strong assumption which means that all the base function \( f_i \) are similar on the sample.

If we apply Theorem 4.3 with the theorem in [37], we obtain the following upper bound on the generalization error.

**Corollary 4.4** Let \( P(x, y) \) be a distribution over \( \mathcal{X} \times \{-1, 1\} \), and let \( S^m \) be a sample of \( m \) examples chosen independently at random according to \( P(x, y) \). Suppose the base-classifier space \( \mathcal{F} \) has VC-dimension \( d \), and let \( \delta > 0 \). Assume that \( m \geq d \geq 1, f \in \mathcal{F}_2 \) where \( \mathcal{F}_2 \) is defined as in Theorem 4.3 and \( \langle f_i, f_j \rangle_{\ell_2} \geq \beta^2 \) where \( \beta > 0 \). Then with probability at
least $1 - \delta$ over the random choice of the training set $S_m$, every weighted average function $f$ satisfies the following bound for all $\gamma > 0$:

$$R_P[f] \leq R_{\text{emp}}^\gamma[f, S_m] + O\left(\sqrt{\frac{dV^2 \log^2 (m/d)}{m\beta^2 \gamma^2}} + \frac{\log(1/\delta)}{m}\right).$$

(4.46)
Chapter 5

Discussion and Conclusions

5.1 Summary

The thesis consists three main parts. We will summarize them separately in this section.

Firstly, we have derived a simplified bound on the covering number of support vector machines based on the result in [59]. This bound is qualitatively different from those that follow from VC theory. It proves that the kernel has a direct influence on the covering numbers. By an example, we have also shown that the bound can be considerably smaller than existing results that did not take account of the kernel. Furthermore, there is an important variable in the bound, which is named as the "effective dimension" of the decision function class. The "effective dimension" can be considered analogous to existing "scale-sensitive" dimensions, such as the fat-shattering dimension follows from VC theory. The difference is that it explicitly depends on the kernel, thus illustrating the character of kernel expansions clearly. For a smooth kernel, the "effective dimension" is small, thus leads to better bounds on generalization performance. This coincides with the idea that a smoother kernel should somehow reduce the capacity of the learning machine.

In the second part, we applied support vector algorithms to two time series prediction problems. The experiment results show good performance. For the mobile
radio signal prediction, which is normally solved by linear methods, support vector algorithms also showed strong performance when some nonlinear kernels were used.

Thirdly, following the successful application of kernel methods in support vector algorithms, we applied this technique to boosting methods. The popular AdaBoost algorithm, which minimizes the sample average of an exponential cost function of the training margins, suffers from overfitting in high noise situations. We have developed some new algorithms, collectively named NormBoost, by adding a regularization term to the standard AdaBoost algorithm. NormBoost approximates smoother functions than AdaBoost. Especially, we designed a RKHS NormBoost, where a function defined in a reproducing kernel Hilbert space (RKHS) is chosen as the regularization term. A particular advantage of the RKHS based regularization is that the optimal linear combination of weak hypotheses can actually be written using only $m$ terms, where $m$ is the number of examples. Thus we need only perform gradient descent in an $m$-dimensional space rather than effectively in an infinite dimensional one, as for the standard AdaBoost algorithm. This leads to a much faster convergence. Experimental results have demonstrated that these algorithms generally outperform AdaBoost with a marked improvement in convergence rate.

5.2 Further work

Based on the thesis, there are a number of open problems and suggestions for future research, which are listed below:

Model selection of support vector machines The new bound we presented for the covering numbers of support vector machines is directly in terms of the eigenvalues of the integral operator induced by the kernel. In our example, we used Gaussian radial basis function as kernels, and achieved the corresponding bound of the covering numbers of the system. The result shows that the width of the kernel influences the covering number. When the width increases, the kernels are smoother, thus the covering number bound decreases. Since the main result shows that changing kernels
can affect the capacity of the function class, and gives the relationship between kernels and the capacity, it seems likely that one can select the suitable model order by using the parameterized family of kernel functions.

**Support vector classification and regression in complex space**  For the experiments of time series prediction in the complex space, we performed SVC or SVR on the real and imaginary part of the training sample separately. Although the experimental results show good performance of the support vector algorithms with nonlinear kernels, there is probability to use other suitable cost functions, such as functions which are directly defined in the complex space. The choices for cost functions in complex valued time series prediction problem is an open problem.

**About NormBoost**  As a combined classification algorithm, NormBoost performances better than normal boosting algorithms. Theorem 1.12 gives the generalization error bound of combined classifiers. How to provide tight bound of NormBoost is an important problem.

The main idea of NormBoost algorithm is to add in a regularization term defined in an inner product space. This opens up the possibility of applying NormBoost to many other machine learning problems.
Bibliography


