Technical University of Denmark

Department of Informatics and
Mathematical Modelling

Kernel Machines for Phoneme Classification

Jesper Salomon

Degree in
Master of Engineering

May 2002

Supervisors:
Lars Kai Hansen
Miles Osborne
and Simon King

Abstract

In this thesis, kernel machines are applied to the problem of phoneme classification: Given a sequence of acoustic observations and 40 phoneme targets, the task is to classify each observation to one of these targets.

To do this, several hurdles must be overcome: Firstly, since this task involves multiple classes, a way to extend the inherently binary kernel machines to handle multiple classes must be found. Secondly, since almost all speech recognition systems are based on statistical models, a probabilistic interpretation of the predictions must be made possible if kernel machines are to be used in a full speech recognition system. Finally, the scaling issues involved in working with large scale problems must be addressed.

The thesis is split up into two parts. The first part examines the Support Vector Machines (SVM), which is the most popular existing kernel method. The chapters in part one focuses on finding effective binary-to-multiclass mappings and examines ways to achieve good performance on a difficult large scale problem. The second part looks for alternative kernel machine that may solve some of the drawbacks found in SVMs. It introduces the Kernel Fisher Discriminant (KFD), which claims to solve the most severe problem with SVMs: the problem of scaling the classifier to handle larger data sets. This claim, along with others, are examined in experiments and compared with SVMs.

Experiments on a well-known speech corpus lead to a best accuracy of 71.4% with the SVM method. Using 14 times fewer components in the model, the KFD achieved a prediction accuracy of 70.2%. This is competitive with the best results found in literature.

A paper based on parts of the work has been submitted to ICSLP 2002.

Acknowledgements

First and foremost, I would like thank my supervisors Simon King and Miles Osborne (Edinburgh University, UK) for help and guidance throughout the project period. Thanks also to my supervisor at the Technical University of Denmark, Lars Kai Hansen.

In addition, I am deeply grateful to Sebastian Mika (GMD First, Germany) for software and advice for the Kernel Fisher Discriminant, and to Tony Van Gestel (Leuwen, Belgium) for help and fruitful discussion regarding the Least Squares SVM.

Finally, thanks to Ronan Collobert (Universite de Montreal, Canada and IDIAP, Switzerland) for the SVMTorch software, and to Thorsten Joachims for providing the SVMLight software.

Contents

Chapter 1

Introduction

The task of this thesis is to create learning machines that can classify sequences of acoustic observations. Consider the situation shown on figure 1.1. Given an ordinary spoken English sentence (labelled (a) on figure), this sentence is split up into tiny fragments of speech, also known as *frames* ((b) on figure). These are converted into vectors of fixed length (c), and fed in to the learning machine (d). It is the goal of the learning machine to classify each of these vectors to one of 40 targets, representing the phonemes in the English language (e).

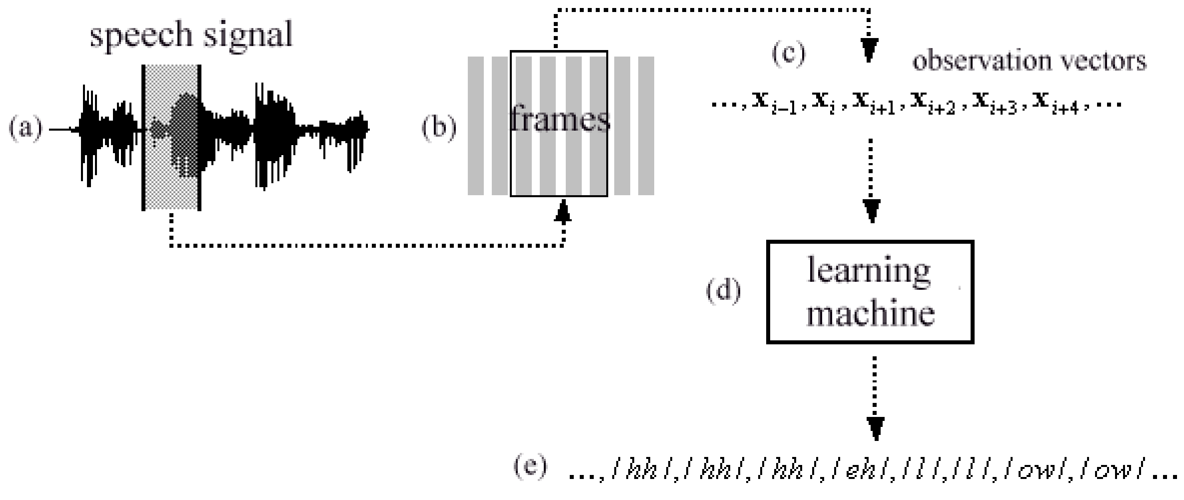

Figure 1.1: Overview of frame classification.

The idea of classifying frames is widely used in both isolated and continuous speech recognition, and is known as *frame-by-frame* classication. The predictions found by the learning machine can be passed on to a statistical model, to find the most likely sequence of phonemes that construct a meaningful sentence. Hence, the classification of frames can be seen as one of the basic units in a speech recognition system[1].

In this thesis, the chosen learning machines for the task are Support Vector Machines (SVMs) and the Kernel Fisher Discriminant (KFD). Both belong to the group of machines called *kernel machines* (explained below).

1.0.1 Support Vector Machines

SVMs was the first kernel machine to be introduced, and is a fairly new technique for pattern recognition, created by Vapnik ([119]). It has been applied to many different problems, and have been very successful in areas such as face recognition ([89]), text-categorisation ([55] and [21]), time-series prediction ([84]), and hand-written digit recognition ([101]). In many of these areas, SVMs have shown to out-perform well-established methods such as Neural Networks and Radial Basis Functions.

This thesis only considers the use of SVMs for pattern *classification*, although they have also been applied to other areas such as Regression ([106]) and Novelty detection ([112]).

In classification, SVMs are binary classifiers. They are used to build a decision boundary by mapping data from the original input space to a higher dimensional feature space, where the data can be separated using a linear hyperplane (see figure 1.2).

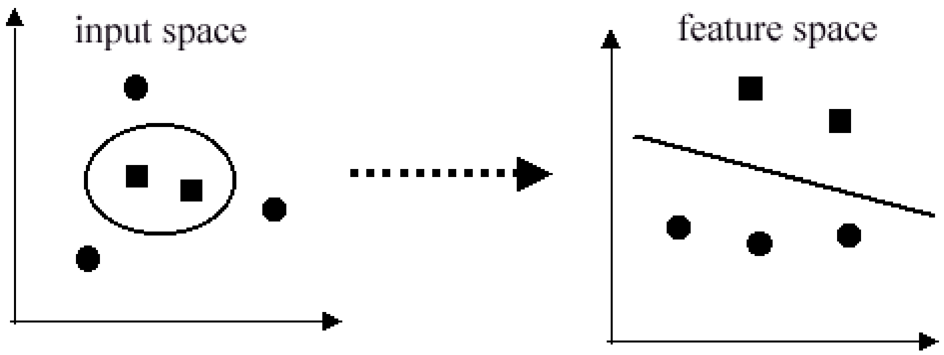

Figure 1.2: The mapping of non-linearly separable training samples from input space to a linearly separable representation in feature space.

Instead of choosing a hyperplane that minimises the training error, SVMs choose the hyperplane that *maximises the margin of separability* between the two sets of data points. This selection can be viewed as an implementation of the *Structural Risk Minimisation* principle, which seeks to minimise the upper bound on the generalisation

[1]Naturally, there are recognition systems that does not use this approach.

error ([119]). Typically, an increase in generalisation error is expected when constructing a decision boundary in higher-dimensional space. By maximising the margin, this expected degradation of performance, also referred to as the *curse of dimensionality*, is counteracted.

The selection of hyperplane in feature-space requires SVMs to evaluate scalar inner products in feature-space. This can be very complicated and computationally expensive if the feature-space is of much higher dimension than input-space. Fortunately, the explicit calculation is not needed due to a functional representation termed a *kernel*. The kernel calculates the inner product in feature-space as a direct operation of the training samples in input-space. Using an effective kernel, this can be done without any significant increase in computational cost.

1.0.2 New Kernel Machines

SVMs belong to the group of learning machines known as *kernel machines*. These all make use of kernels to map data from the original input space to a higher dimensional feature space (as explained above). This enables them to create non-linear separations in input space. Numerous new kernel machines have been introduced after the success of SVMs[2]. It is not feasible to even mention them all, but among the most successful ones are the Relevance Vector Machine [113], Gaussian Processes [73, 122], Least Squares Support Vector Machines [110], Bayes Point Machines [49], Boosting methods [2], and the Kernel Fisher Discriminant [80].

The second part of the thesis focuses on selecting a new kernel machine, which can improve upon the results found by SVMs in part I. Several methods are reviewed, and one method is chosen, examined in details and compared with SVMs.

1.0.3 The Kernel Fisher Discrimiannt

The chosen method to be examined in the second part is the Kernel Fisher Discriminant (KFD). The method is a variation of Fisher's classical linear discriminant [28] that, as SVMs, finds a decision boundary by mapping data from the original input space to a higher dimensional feature space, where the data can be separated using a linear hyperplane (as seen figure 1.2).

However, the way the hyperplane is chosen is different from SVMs. Whereas SVMs attempts to maximise the (minimum) margin, the KFD looks at the *distribution* of the data. I.e. it bases the choice of decision boundary by looking at class means and class variances.

[2]The website http://www.kernel-machines.org lists hundreds of papers dealing with new kernel methods.

More specifically, to separate the classes the KFD uses two factors: a) the distance between the means of the two classes (should be maximised), and b) the within-class variance of each class (should be minimised). For this reason, the KFD can also be said to *maximise the average margin* between the classes. Using these two criteria to separate the classes have proven to create a very successful classifier [80].

1.0.4 Hurdles to overcome

By using the KFD and SVMs in speech recognition, it is hoped that some of the performance gains shown by kernel machines on other tasks might be mapped over to this domain.

Limited work has been done to apply the methods to the problem, and most have involved only SVMs. The results so far have been promising, but several significant hurdles must be overcome if kernel machines are to be viewed as a viable and competitive alternative to the established methods. Problems include:

- *The problem of multi-class classification.* Both KFD and SVMs are inherently binary classifiers. A method for effectively extending the binary classification to the multi-class case is needed. This method must be able to retain the good generalisation ability of the binary classifier.

- *Estimation of posterior probabilities.* Most speech recognition systems are based on statistical models that combine low-level acoustic information with higher-level knowledge of the language. For any classifier to be used in such a system, it needs to be able to return class conditional probabilities.

- *Choosing an appropriate kernel.* The kernel affects the performance of the classifier. A method for choosing and parametrising the kernel is needed.

- *Working in a real-time environment.* Since most speech recognition systems are created for real-time communication, the complete system must be able to process the spoken data as it is dictated. For a learning machine to function in such an environment, it must be extremely effective.

I have chosen to focus on the first three of the four problems. Even though having a fast and effective method is important, I feel it is more important to examine the feasibility of applying kernel machines to the task without being concerned about speed. Other methods have been introduced in a similar fashion. Hidden Markov Models (HMMs), the method used in most commercial speech recognition systems, would be hideously slow if it was used without any optimisation techniques. It is because of techniques for quickly removing unlikely predictions early on in the recognition process that makes HMMs suitable for real-time recognition[3].

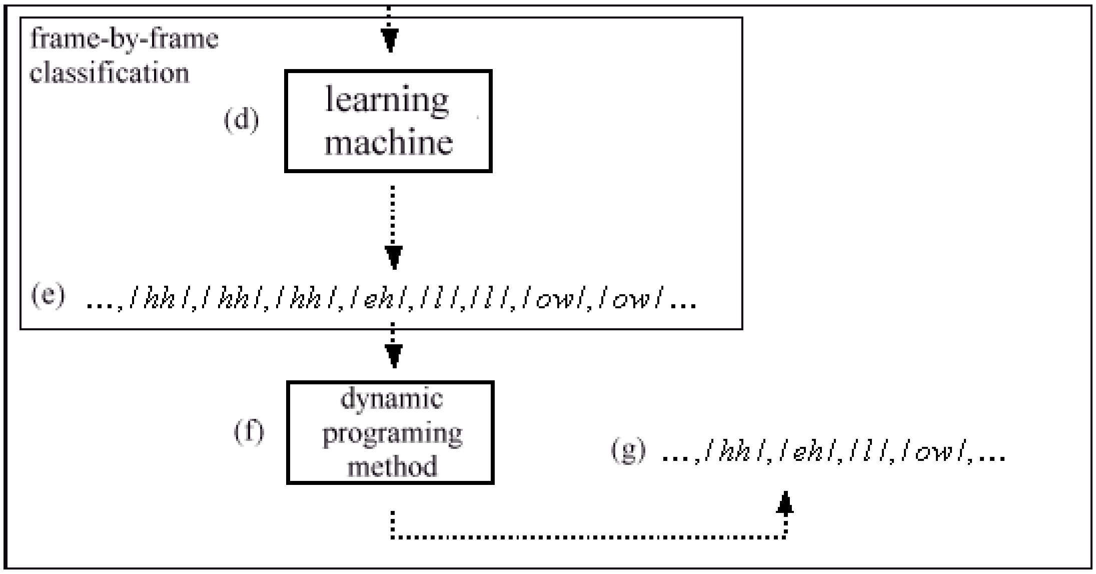

Figure 1.3: Difference between phoneme recognition and frame-by-frame classification. The small box encloses the task of a frame-by-frame classifier, and the big box encloses the task of a phoneme recogniser. After the frames have been classified in (e), the dynamic programming method (f) finds the most likely sequence of phonemes (g).

The task of *frame-by-frame* classification is typically done in context of *phoneme recognition.* As the name implies, phoneme recognition consists of identifying the individual phonemes a sentence is composed of. This differs slightly from *frame-by-frame* classification in the sense that phonemes typically have much longer durations[4]. Hence, one phoneme spans several frames, and a dynamic programming method is required to transform the frame classifications into phoneme predictions. This difference is shown on figure 1.3.

I have chosen not to perform complete phoneme recognition, because I would need to include a dynamic programming method. This may seem fairly straightforward, but to do it properly, it would involve introducing large areas of speech recognition such as decoding techniques and the use language models. This would detract from the focus of the thesis. Furthermore, by concentrating on the task of *frame-by-frame* classification, there will be no other methods to potentially "distort" the performance of the kernel machine classifier.

[3]The optimisation techniques are better known as *pruning*.

[4]Typically, *frames* are extracted with 10 ms intervals. On average, phonemes lasts 65ms ([43]).

1.1 Summary of recent literature

Various techniques have been applied to the problem of *frame-by-frame* classification. This section will first review Recurrent Neural Networks. They are a method that has lead to results substantially better than any other results reported in literature. Subsequently, the few previous attempts to apply SVMs to the task will be described. These will be reviewed with reference to the three hurdles described above.

1.1.1 Recurrent Neural Networks

In 1991, Robinson ([94]) created a phoneme recognition system that has produced the best phoneme recognition results reported in literature[5]. He improved the system over the years, and in 1994 he reported a phoneme accuracy of 80%, and a frame-by-frame accuracy of 70.4%[6]. Until recently, this was the best-reported result on the task. The system utilized a time-delayed recurrent neural network (RNN), a variant of the standard neural networks specifically invented for the task. In contrast to a typical Neural Network, predictions are time-delayed so future frames are considered before a decision is made. Moreover, previous predictions are fed back into the network to add context without having to feed multiple frames of data to the classifier. Combined with numerous metrics for evaluating predictions, this created an effective classifier for handling segmented speech data. It was extended to incorporate knowledge of language, and currently competes head-to-head with commercial speech recognition packages.

In 1998, Chen and Jamieson reported results based on an extensive set of experiments ([13]). They used an almost identical approach to Robinson's RNN, but included a new criterion function that enabled them to directly maximise the frame classification accuracy. Even though their phoneme recognition rate of 79.9% was slightly worse than Robinson's result, their best frame classification result was 74.2%.

1.1.2 Kernel Machines in Speech Recognition

To date, only SVMs have been applied to the task of speech recognition.

In [17], Clarkson created a multi-class SVM system to classify phonemes[7]. He assumed phoneme boundaries to be known, and mapped the variable length phonemes to fixed length vectors. This was done such that the most of the important information

[5] The project involved full phoneme recognition. However, frame classification results were also reported.

[6] This result was found by mapping the 61 phoneme targets used by Robinson down to the 40 phoneme targets used in this thesis (thanks to Simon King for providing this result).

[7] The system was a hybrid between frame-by-frame classification and phoneme recognition.

was retained, and thus the problem was considerably easier than that of frame-by-frame classification[8]. Still, the reported result of 77.6% is extremely encouraging and shows the potential of SVMs in speech recognition. Additionally, the multi-class classifier used in the paper proved that it is possible to successfully extend the generalisation performance of the binary SVM to the multi-class case.

In [33] and [32], a hybrid SVM/HMM system was developed. The SVMs were used to generate predictions that were fed to the HMM. Similarly to the above paper, the classification task of the SVM was to classify phonemes where the phone boundaries were known. For this, the same mapping was used as in [17] explained above. The performance of the system was compared with a standard HMM system where Gaussian Mixture Models (GMMs) were used instead of SVMs. On the OGI Alphadigits speech database ([18]), the SVM/HMM outperformed the GMM/HMM system by a relative 10%. Attractively, only a fifth of the training data were used in the SVM/HMM hybrid compared to that of the GMM/HMM system. Additionally, the resulting SVM/HMM system had an order of magnitude less free parameters than the GMM/HMM system.

In [85], data-dependent kernels were created to handle dynamically length sequences. A generative probability model were run on the variable length phonemes to produce fixed length "scores" for each sample. Sequently, SVMs classified the samples according to their "score values". As a first step to enable SVMs to handle variable length sequences, the results were encouraging.

In [15], the practical issues of training SVMs in the context of speech recognition were examined. Problems with non-converging training algorithms were solved, and two multi-class systems were created and shown to produce good results on a handpicked subset of the TIMIT speech corpus ([34]).

In addition to recognition, SVMs have also been used in other speech and sound applications. These include speaker identification ([98]), musical instrument classification ([76]), utterance verification ([72]), and web audio classification ([83]). These will not be discussed in details, but these are worth mentioning since they all produced promising results.

The above papers all show the potential of using SVMs, and other kernel methods, in speech recognition. Some have discussed and attempted to tackle some of the problems involved, but kernel methods have not yet been applied to a full classification problem where no assumptions are made about the speech data. Furthermore, most of the above papers created their own benchmark systems. Since these were not the main focus of the papers they may may not have been completely optimised, and hence their results may have been sub-optimal. It is the focus of this thesis to evaluate the

[8] The result cannot be compared with the results of the RNNs from the previous section

kernel methods on a well-known and recognised classification task, to allow a direct comparison with the best results found in literature.

1.1.3 Overview of thesis

The rest of the thesis is split up into two parts:

Part I: Support Vector Machines In chapter 2, I explain the concepts behind linear and non-linear SVMs, feature space, and the role of the kernel. In addition, I describe two techniques for extending the inherently binary SVMs to multi-class problems.

In chapter 3, experiments using the two multi-class methods will be presented. The experimental results will be discussed and compared with the best results found in literature.

In chapter 4, I extend the SVM classification process, to enable SVMs to return posterior probabilities. In addition, a new multi-class method is introduced to take advantage of the extension.

Part II: New Kernel Machines In chapter 5, I review alternative kernel machines that can potentially improve upon the results found by the SVM in chapter 3.

In chapter 6, I introduce the chosen kernel machine: the Kernel Fisher Discriminant. I explain the ideas behind it and describe a technique for speeding up the training process.

In chapter 7, I report results from experiments performed with the KFD method. The results are discussed and compared with the ones found by the SVM.

Finally, chapter 8 will summarise findings and give directions for possible future research.

As a last note, Appendix C contains a paper submitted to ICSLP 2002[9]. Since it gives a brief 4-page summary of parts of the project, it might provide a useful overview.

[9]"International Conference for Spoken Language Processing" in Denver, US.

Part I: Support Vector Machines

Chapter 2

Support Vector Machines

2.1 Introduction

In this chapter, the Support Vector Machine formulation for pattern classification will be described[1]. The formulation defines SVMs as a maximum margin classifier, which implements the *Structural Risk Minimisation* principle. This principle seeks out to minimise the upper bound of the generalisation error.

The chapter will start by explaining the statistical learning theory upon which SVMs, and many other kernel machines, are based. Secondly, the advancement of SVMs from being a linear maximum margin classifier to a non-linear classifier will be carefully described. Finally, I will explain some common methods for extending the binary SVMs to handle multiple classes.

2.2 Statistical Learning Theory

This chapter describes the statistical theory upon which SVMs are based. The theory is not only appliable to SVMs, but forms the basis for both kernel machines described in this thesis.

In general, SVMs seek to solve a binary classification problem. In this section, the statistical foundation behind the technique will be described in the context of such a problem. Moreover, the theory will be compared to what is employed in other learning machines

[1]This chapter is based on [96], which, in turn, is based on [11], [123], [15], and [40].

2.2.1 Binary Classification Problem

The general two-class classification problem can be stated as follows.

1. Given a data set D of N samples: $\langle \mathbf{x}_1, y_1 \rangle, \langle \mathbf{x}_2, y_2 \rangle, \ldots, \langle \mathbf{x}_N, y_N \rangle$. Each sample is composed of a training example $\mathbf{x}_i$ of length M, with elements $\mathbf{x}_i = \langle x_1, x_2, \ldots, x_M \rangle$, and a target value $y_i \in \{-1, +1\}$.

2. The goal is to find a classifier with decision function, $f(\mathbf{x})$, such that $f(\mathbf{x}_i) = y_i$, for all samples $\langle \mathbf{x}_i, y_i \rangle \in D$.

The performance of such a classifier is measured in terms of the classification error defined in equation (2.1).

$$error(f(\mathbf{x}), y) = \begin{cases} 0 & \text{if} f(\mathbf{x}) = y \\ 1 & \text{otherwise} \end{cases} \tag{2.1}$$

2.2.2 Empirical Risk Minimisation

Consider a learning machine with a set of adjustable parameters $\boldsymbol{\beta}$. Given the above binary classification task, the machine seeks to find $\boldsymbol{\beta}$ such that it learns the mapping $\mathbf{x} \mapsto y$. This will result in a possible mapping $\mathbf{x} \mapsto f(\mathbf{x}, \boldsymbol{\beta})$ that defines the machine. The performance of the machine is measured by the empirical risk error:

$$R_{emp}(\boldsymbol{\beta}) = \frac{1}{N} \sum_{i=1}^{N} error(f(x_i, \boldsymbol{\beta}), y_i) \tag{2.2}$$

where N is the size of the training set and $\boldsymbol{\beta}$ the set of adjustable parameters.

This risk minimisation principle is called *Empirical Risk Minimisation* (EMP). Most learning machines implements this principle, including Neural Networks and most instance-based methods. This has lead to many efficient and effective classifiers.

However, there is one problem with these machines. If the complexity of a machine is high, it has a tendency to *overfit* the data[2]. This is because the minimisation principle does not consider the *capacity* of the machine, i.e. how many different, or complex, mappings a machine can make.

[2] Given a set of training samples D and a separate set of test samples G (from the same distribution). *Overfitting* occurs when the learning machine learns a mapping $\mathbf{x} \mapsto y$ such that it classifies the set D better than it does G.

2.2.3 Structural Risk Minimisation

In contrast to EMP, the *Structural Risk Minimisation* principle (SRM) considers the complexity of the learning machine when it searches for $\boldsymbol{\beta}$ to learn the mapping $\mathbf{x} \mapsto y$. This is done by minimising the *expected risk*:

$$R_{\exp}(\boldsymbol{\beta}) = \int error(f(x_i, \boldsymbol{\beta}), y_i) dP(x, y) \tag{2.3}$$

where $P(x, y)$ is a prior probability.

Unfortunately, we cannot explicitly calculate the expected risk since $P(x, y)$ is unknown in most classification tasks. Instead, an estimate of this risk is used. [120] showed that the upper bound on this error is:

$$R_{\exp}(\boldsymbol{\beta}) \leq R_{emp}(\boldsymbol{\beta}) + VC(H) \tag{2.4}$$

where $VC(H)$ is the Vapnik Chervonenkis (VC) dimension (explained below).

Hence, we can estimate the risk by computing this upper bound. This is done by calculating R_{emp} from the training data, and estimating $VC(H)$ of the learning machine.

2.2.4 VC dimension

The VC dimension is a measure of the *capacity* of a learning machine. By definition, it is the largest set of points X that can be *shattered* by the learning machine. To explain what *shattered* means, consider the training set X. If a learning machine with mapping $\mathbf{x} \mapsto f(\mathbf{x}, \boldsymbol{\beta})$ can correctly assign all possible labels to the instances in X, then this learning machine is said to *shatter* X. Consequently, the VC dimension is closely related to the complexity of the learning machine.

Learning machines with a large number of free parameters, and therefore a large VC dimension, are generally expected to have a low empirical risk since they can model more complex decision surfaces[3]. However, based on the above formula of the expected risk, we can also conclude that the confidence in the empirical risk decreases if the complexity of the learning machine increases. This relation is shown in figure 2.1. SRM can therefore be described as a trade-off between the *quality* of the approximation of the data and the *complexity* of the approximating function [119].

How SVMs are constructed to implement the SRM is described in the following sections.

[3] This is not always the case. Some learning machine with a high number of free parameters can have a low VC dimension, and vice-versa.

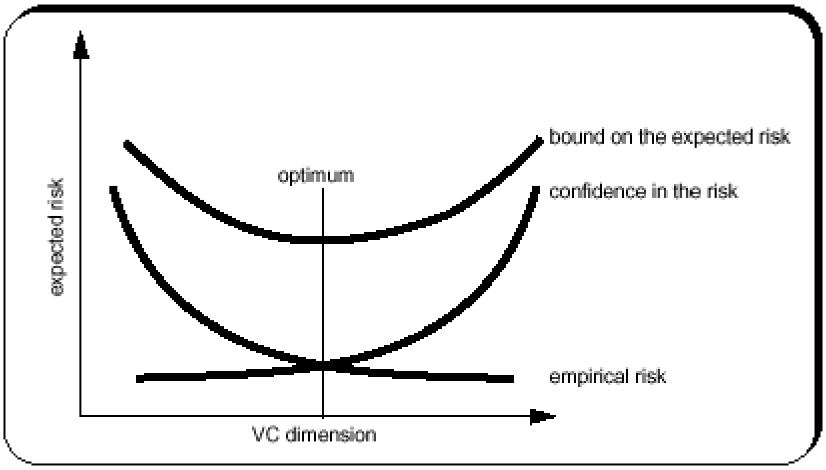

Figure 2.1: Relation between empirical risk, VC dimension and expected risk (from [32]).

2.3 Linear Classifiers

In the following two sections, SVMs will be formulated as a maximum margin classifier[4]. There are two cases of linear classifiers to consider. In section 2.3.1, the case where a perfect mapping $\mathbf{x} \mapsto f(\mathbf{x}, \beta)$ can be learned will be explained. Subsequently, in section 2.3.2, the case where a perfect mapping is unattainable is described.

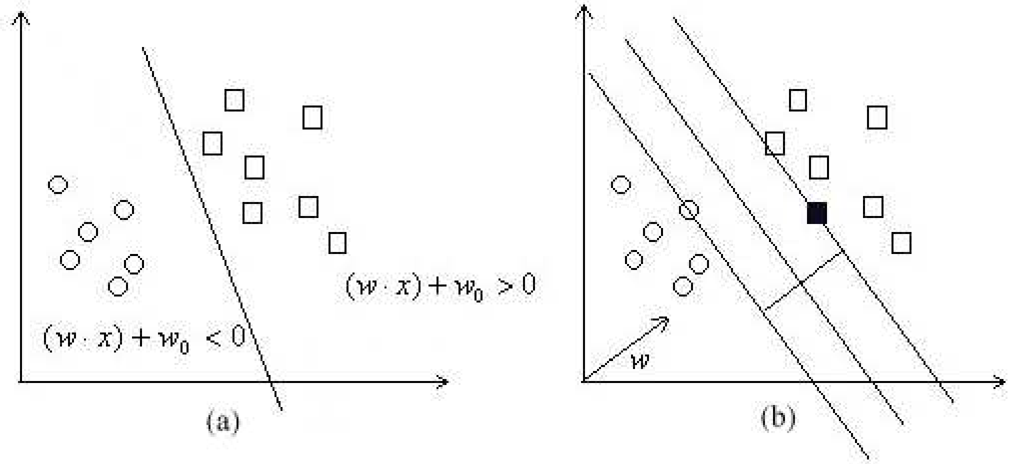

Figure 2.2: (a) a separating hyperplane. (b) The hyperplane that maximises the margin of separability

[4]From now on the learning machine will be called a *classifier*.

2.3.1 The Separable Case

Consider the binary classification problem of an arrangement of data points as shown in figure 2.2(a). We denote the "square" samples with targets $y_i = +1$ as *positive* examples, belonging to the set S_+. Similarly, we define the "round" samples with $y_i = +1$ as *negative* examples, belonging to S_-.

One mapping that can separate S_+ and S_- is:

$$f(\mathbf{x}, y) = sign(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.5}$$

where $\mathbf{w}$ is a weight vector and b the offset from origin. This pair $\{\mathbf{w}, b\}$ is the SVMs' version of the parameter set $\boldsymbol{\beta}$ from the section 2.2.2.

Given such a mapping, the hyperplane

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2.6}$$

defines the decision boundary between S_+ and S_-. The two data sets are said to be *linearly separable* by the hyperplane if a pair $\{\mathbf{w}, b\}$ can be chosen such that the mapping in equation (2.5) is perfect. This is the case on figure 2.2(a), where the "round" and "square" samples are clearly separable.

A Maximum Margin Separating Hyperplane

There are numerous values of $\{\mathbf{w}, b\}$ that can create separating hyperplanes. The SVM classifier finds the only hyperplane that maximises the margin between the two sets. This is shown in figure 2.2(b). The data sets are said to be optimally separated when such a boundary is found.

To find the optimal hyperplane, notice how equation (2.5) is a discriminant function. Therefore, we can scale the hyperplane by k to $k(\mathbf{w} \cdot \mathbf{x} + b)$, and still have the same discriminant function. Hence, we can choose to scale $\mathbf{w}$ and b as we like. If we choose a scaling such that $|f(\mathbf{x}_{near}, \{\mathbf{w}, b\})| = 1$, where $\mathbf{x}_{near}$ is the training example nearest the decision plane, we get:

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i - b \geq +1 \quad for \quad y_i = +1 \\ \mathbf{w} \cdot \mathbf{x}_i - b \leq -1 \quad for \quad y_i = -1 \end{aligned} \tag{2.7}$$

Or written in a more compact representation:

$$y_i\left(\mathbf{w} \cdot \mathbf{x}_i - b\right) \geq +1 \quad \forall i \in D \tag{2.8}$$

Looking at figure 2.2(b), if we define the margin as $d_+ + d_-$ it is obvious that for the margin to be maximal, d_+ must be equal d_-. Moreover, consider the positive training example $\mathbf{x}_+ \in S_+$ with the shortest perpendicular distance d_+ from the separating hyperplane (marked as a black "square" on figure 2.2(b)). This example will lie on the hyperplane $\mathbf{x}_+ \cdot \mathbf{w} + b = 1$ and satisfy the equality in equation (2.8). Similarly, we can find a negative training example $\mathbf{x}_- \in S_-$ that satisfies $\mathbf{x}_- \cdot \mathbf{w} + b = -1$. We denote these hyperplanes H_+ and H_-.

The margin between the hyperplanes, $d_+ + d_-$, can be reformulated as:

$$\begin{aligned} d_+ + d_- &= \tfrac{|\mathbf{w}\cdot\mathbf{x}_+ + b|}{\|\mathbf{w}\|} + \tfrac{|\mathbf{w}\cdot\mathbf{x}_- + b|}{\|\mathbf{w}\|} \\ &= \tfrac{1}{\|\mathbf{w}\|}\left(|\mathbf{w}\cdot\mathbf{x}_+ + b| + |\mathbf{w}\cdot\mathbf{x}_- + b|\right) \\ &= \tfrac{2}{\|\mathbf{w}\|} \end{aligned} \tag{2.9}$$

H_+ and H_- both have the same normal $\mathbf{w}$, and are therefore parallel. Since no training points fall between them, the hyperplane that optimally separates the data is the one that minimises $\|\mathbf{w}\|^2$, subject to constraints in equation (2.8).

Notice, that this solution is independent of the bias b. Changing b will move the optimal hyperplane in the direction of $\mathbf{w}$ and the maximum margin will remain the same. However, the separating hyperplane will no longer be optimal since it will be nearer to one of the classes ($d_+ \neq d_-$).

Structural Risk Minimisation and the optimal hyperplane

To explain how the SRM principle is implemented by maximising the margin of separability, suppose an upper bound on $\|\mathbf{w}\|^2$ exist:

$$\|\mathbf{w}\| \leq A \tag{2.10}$$

From equation (2.8) we then get:

$$d_i \geq A, \forall i \in \{+, -\} \tag{2.11}$$

So the separating hyperplanes *cannot be closer than* $1/A$ *to any data points in the two sets.*

Now, consider the arrangement of data points in figure 2.3(a). The number of possible separating hyperplanes is shown. Then consider the situation in figure 2.3(b). If no hyperplane can be closer than $1/A$ from the data points, this reduces the number of possible separating hyperplanes.

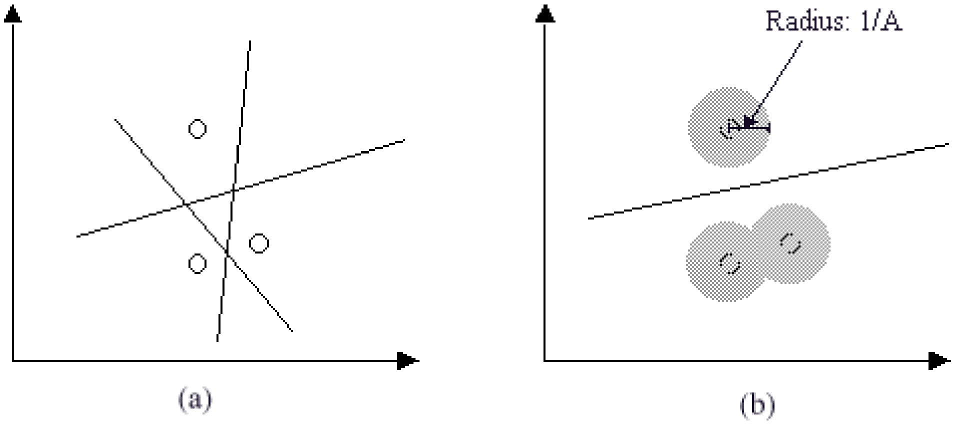

Figure 2.3: (a) All hyperplanes that separates the data points are shown. (b) The number of possible separating hyperplanes is reduced when an upper bound A is put on $\|\mathbf{w}\|$.

From [119], the VC dimension $VC(H)$, for a classifier that can find all such separating hyperplanes when an upper bound is put on $\|\mathbf{w}\|$, is:

$$VC(H) \leq \min(R^2A^2, N) + 1 \tag{2.12}$$

where A is the upper bound on $\|\mathbf{w}\|$, N the number of dimensions of the data points, and R is the radius of the hyper sphere enclosing all data points.

In conclusion, a smaller $\|\mathbf{w}\|$ reduces the number of separating hyperplanes. Consequently, minimising $\|\mathbf{w}\|^2$ is equal to minimising the upper bound of the VC dimension.

Lagrangian formulation

Finding an optimal separating hyperplane involves solving a minimisation problem of $\|\mathbf{w}\|^2$with a set of inequality constraints from from equation (2.7)[5]. The theory of Langrangian multipliers is well known to efficiently solve this problem ([10]).

The Lagrangian formulation of the minimisation problem is:

$$\text{Minimise} \left\{ L_P \equiv \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^{N} \alpha_i y_i \left(\mathbf{x}_i \cdot \mathbf{w} - b \right) + \sum_{i=1}^{N} \alpha_i \right\} \tag{2.13}$$

Subject to constraints:

[5]The reason why the L2-norm is specifically chosen in $\|\mathbf{w}\|^2$ is because elegant techniques exist to optimise convex functions with constraints.

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{2.14}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.15}$$

This is called the *primal* formulation. It is a convex quadratic programming problem because the objective function L_p itself is convex.

Since the constraints are equality constraints, the *dual* formulation can be found by substituting the constraints into the objective function. The resulting dual formulation is:

$$\text{Maximize} \left\{ L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j \right\} \tag{2.16}$$

Subject to constraints:

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.17}$$

In conclusion, SVM training can be considered as a problem of maximising L_D with respect to α_i, subject to constraints (2.17) and positivity of α_i. The resulting optimal hyperplane is given by:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{2.18}$$

and bias b:

$$b = -\frac{1}{2} \mathbf{w}(\mathbf{x}_+ + \mathbf{x}_-) \tag{2.19}$$

This solution is a *linear combination* of the $\mathbf{x}_i$s. An important detail is that $\alpha_i = 0$ for every $\mathbf{x}_i$ *except* the ones that lie on the hyperplanes H_+ and H_- . These points, where $\alpha_i \neq 0$, are called *Support Vectors*. Hence the name *Support Vector Machines*. The number of support vectors in the solution is typically much less than the total number of training examples. This is referred to as the *sparsity* of the solution.

An interesting observation can be found if all training examples except the support vectors were removed *before* training. In this case, after training the solution would remain the same. Also notice that the quadratic programming problem is convex, i.e. there are no local minima. Consequently, the solution will always find the global minima, and thus be optimal. This gives SVMs an advantage over other optimisation techniques such as Neural Networks, where local minimas exist.

Classification

When we have solved the optimisation problem and found the optimal separating hyperplane, the SVMs can attempt to predict unseen instances. An instance $\mathbf{x}$ is classified by determining on which side of the decision boundary it falls. To do this, we compute:

$$f(\mathbf{x}) = sign(\mathbf{w} * \mathbf{x} + b) = sign\left(\sum_{i=1}^{n} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + b\right) \tag{2.20}$$

and thereby assign it to one of the target labels +1 or –1, representing the positive and negative examples. Note that the input instance $\mathbf{x}$ only enters this function in the form of its inner product. This is exploited later when the extension to the non-linear case is done.

2.3.2 Non-separable case

So far, the SVM formulation has been restricted to the case where a perfect mapping, $\mathbf{x} \mapsto f(\mathbf{x}, \beta)$, can be learned. In general, though, most real-world data sets do not satisfy this condition. An extension of the above formulation to handle non-separable data is needed, and the hyperplane must be found such that the resulting mapping is the best possible.

The extension is done by creating an objective function that trades off misclassifications against minimising $\|\mathbf{w}\|^2$. Misclassifications are considered by adding a "*slack variable*" $\xi_i \geq 0$ for each training example, and require that:

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i - b &\geq +1 - \xi_i \quad for \quad y_i = +1 \\ \mathbf{w} \cdot \mathbf{x}_i - b &\leq -1 + \xi_i \quad for \quad y_i = -1 \end{aligned} \tag{2.21}$$

This adjustment allows the old constraints in equation (2.7) to be violated, but in a way that a violation causes a penalty. The size of the penalty for each misclassified example, the value of ξ_i, is typically the distance from the decision boundary to the training example[6]. On figure 2.4, an overview of a non-separable case is shown.

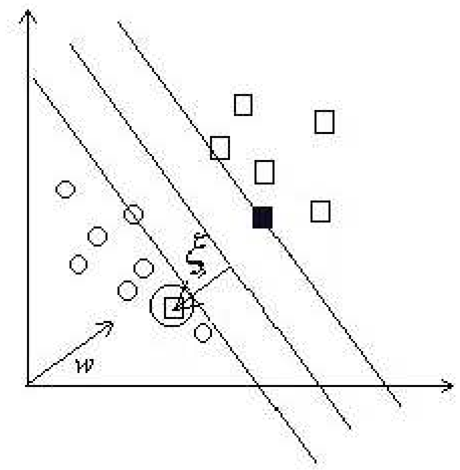

Figure 2.4: A non-separable case. The encircled data point is misclassified and thus has a positive ξ.

The new problem we are faced with is to minimise the sum of misclassification errors as well as minimising $\|\mathbf{w}\|^2$:

$$\|\mathbf{w}\|^2 + C\left(\sum_i \xi_i\right)^k \tag{2.22}$$

where C is a regularisation parameter used to control the relation between the slack variables and $\|\mathbf{w}\|^2$. k is an integer with typical values of 1 or 2.

This minimisation problem is also convex as in the linearly separable case. If we choose k to be 1, it has the advantage that all ξ_i's and their Lagrange multipliers disappear from the dual Langragian problem ([11]). The objective function of the dual formulation becomes:

$$\text{Maximize}\left\{L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \mathbf{x}_i\mathbf{x}_j\right\} \tag{2.23}$$

Subject to constraints:

$$0 \leq \alpha_i \leq C \tag{2.24}$$

[6]Other error measures exist, e.g. the squared distance.

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.25}$$

When optimised, the solution is still given by:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{2.26}$$

and:

$$b = -\frac{1}{2}\mathbf{w}(\mathbf{x}_+ + \mathbf{x}_-) \tag{2.27}$$

where $\mathbf{x}_+$ is the positive example with shortest perpendicular distance from the decision boundary, and $\mathbf{x}_-$ is closest negative example.

To compare this solution to that of the linearly separable case, the only difference is the added constraint (equation (2.24)). Now α_is have an upper bound of C .

The support vectors in this solution are not only the training examples that lie on the hyperplane boundary. It is also the training examples that either falls between the two hyperplanes H_+ and H_- or falls on the wrong side of the decision surface. Hence, one can view the support vectors as modelling the error of the boundary between the two classes.

2.4 Non-linear Classifiers

In the above section, we described how the linear SVM could handle misclassfied examples. Another extension is needed before SVMs can be used to effectively handle real-world data: the modelling of non-linear decision surfaces.

The method for doing this was proposed by [41]. The idea is to explicitly map the input data to some higher dimensional space, where the data *is* linearly separable. We can use a mapping:

$$\Phi : \Re^N \rightarrow \Im \tag{2.28}$$

where N is the dimension of the input space, and $\Im$ a higher-dimensional space, termed *feature space*.

In feature space, the technique described in the above section can be used to find an optimal separating hyperplane. When the hyperplane is found, it can be mapped back down to input space. If the mapping Φ is non-linear, the resulting hyperplane in input-space will also be non-linear.

This technique can be described in a three step process, as shown on figure 2.5.

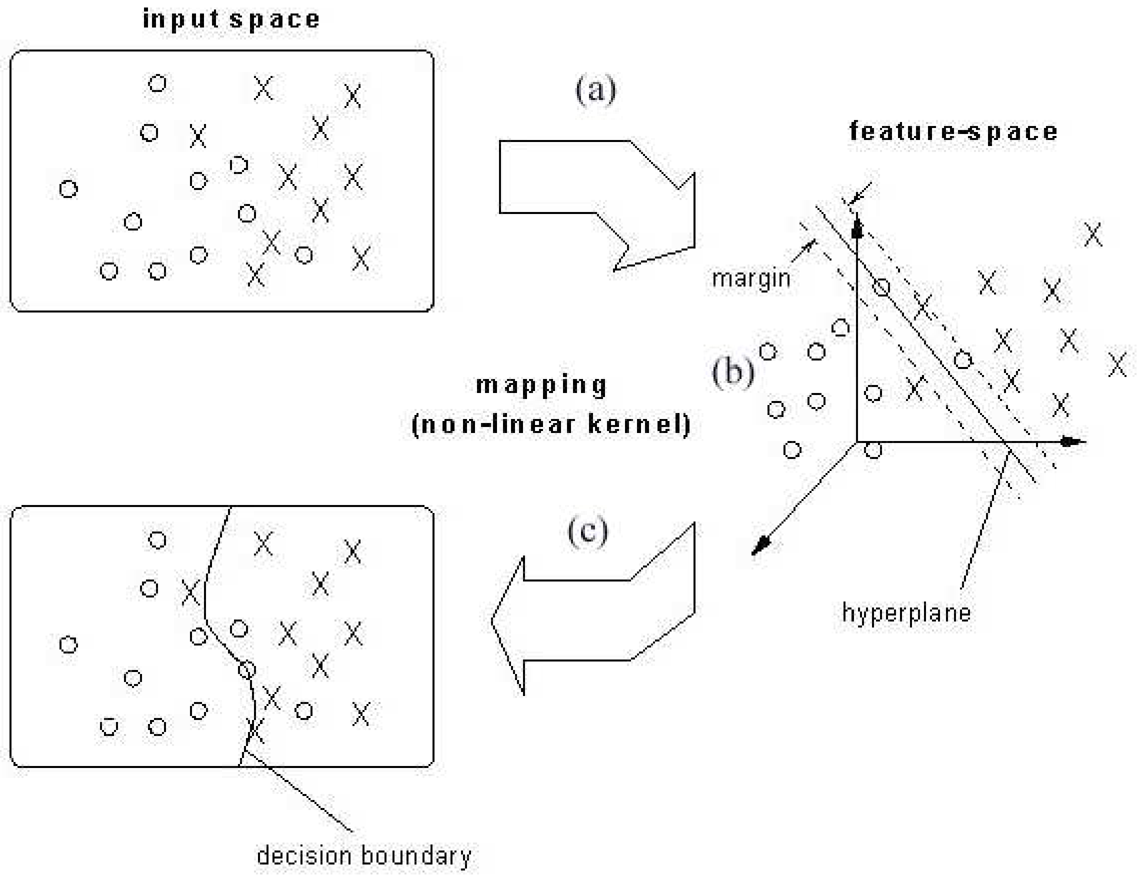

Figure 2.5: The role of the kernel (from [85]). (a) The data is mapped from input-space to feature-space by a mapping Φ. (b) The optimal separating hyperplane is found. (c) The hyperplane is mapped back down to input-space, where it results in a non-linear decision boundary.

2.4.1 The "Kernel Trick"

Most often, finding the optimal hyperplane in the higher dimensional feature space is both complicated and computationally expensive. It was not before Vapnik, Boser and Guyon ([41]) showed that an old trick by [71] can be used, called the "*kernel trick*". Using this trick, the above three steps could be combined into one.

In the training phase described in the previous section, notice equation (2.23) only includes the training data in the form of their scalar inner products $\mathbf{x}_i \cdot \mathbf{x}_j$. Thus, a mapping from input-space to feature-space can be achieved via a substitution of the inner product with:

$$\mathbf{x}_i \cdot \mathbf{x}_j \rightarrow \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

Fortunately, calculating each Φ explicitly is not needed. Instead, we can use a functional representation $K(\mathbf{x}_i, \mathbf{x}_j)$ which computes the inner product in feature space as a direct operation upon the data samples in their original input space:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \tag{2.29}$$

This functional representation is called a *kernel*, and SVMs are a member of the broader class of kernel methods, also termed *kernel machines* [100]. The other classifiers described in part II of the thesis are all kernel machines that uses this trick.

If the feature-space is of much higher dimension than the input space, this implicit calculation of the dot-product performed by the kernel removes the need to explicitly perform calculations in feature space. Consequently, if an effective kernel is used, finding the separating hyperplane can be done without any substantial increase in computational expense[7].

2.4.2 Training

The optimisation problem changes slightly to accommodate the kernel. If we substitute the inner product in equation (2.23) with the kernel function, the new optimisation problem we are faced with becomes:

$$\text{Maximize} \left\{ L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \right\} \tag{2.30}$$

Subject to same constraints as in the non-separable case:

$$0 \leq \alpha_i \leq C \tag{2.31}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.32}$$

[7] In fact, in most cases the extra computational expense is so small that it takes roughly the same time to find a non-linear optimal hyperplane than to find a optimal linear hyperplane.

2.4.3 Classification

Luckily, the same trick can be used in classification. In equation (2.20), where unseen examples $\mathbf{x}$ and support vectors $\mathbf{x}_i$ are only included through their inner product $\mathbf{x}_i\mathbf{x}$. So we can replace $\mathbf{x}_i\mathbf{x}$ with $K(\mathbf{x}_i, \mathbf{x})$ to get:

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \tag{2.33}$$

where b the offset of the decision boundary from origin.

Classification an unseen examples, $\mathbf{x}$, are subsequently done by:

$$g(\mathbf{x}) = sign(f(\mathbf{x})) \tag{2.34}$$

2.5 Kernels

It is not all functions that can be used as kernels. Feasible kernels must satisfy the following two conditions:

1. *The kernel function must be symmetric.*
2. *It must satisfy Mercer's Theorem* ([10]).

These conditions are described in details in Appendix A.

Some popular kernels includes:

- The homogenous polynomial kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d \tag{2.35}$$

  If input-space is $\Re^N$, this kernel maps the samples into a fixed $(N+d)$-dimensional space[8]. The resulting the decision boundary will be polynomial ([11]).

- The Gaussian kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2} \tag{2.36}$$

[8] The value of d is also termed a *kernel-parameter* as it must be defined by the user before training begins.

This kernel centres a Gaussian with variance σ[9] on each support vector. This kernel is also called the *Radial Basis Function* Kernel, since the resulting classifier is closely related to the *Radial Basis Function* (RBF) learning machine ([40]).

- The Linear kernel:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j \tag{2.37}$$

This is a kernel which does not perform any mapping, and the resulting SVM attempts to find a linear decision boundary in input space as in equation (2.23)[10]. Note, that one can also consider this kernel as a specific case of the Polynomial kernel with $d = 1$.

2.6 Feature Space vs. Input space

The extension of the SVM to include kernels is very elegant, and enables it to describe non-linear decision boundaries. The kernel allows computation of scalar products in a high dimensional feature space where it otherwise would be extremely hard to perform *any* calculations. However, the solution is found without knowing the explicit mapping from input space to feature space. Consquently, although it is elegant, the use of kernels are not without drawbacks. It is the difficult to get an understanding of what exactly is going on during training and how the data is distributed in feature space. Several papers have discussed this issue [45, 102, 36]. However, all these accept that a clear understanding of the kernel mapping is rarely possible, unless the kernel is very simple. In the case of the polynomial kernel, figure 2.5 visualises the effect of using a simple 2 degree polynomial kernel. In the case of the Gaussian kernel, we know that it corresponds to centering a Gaussian distribution on each data point. Each support vector will have "influence" on the separation depending on the Gaussian variance σ and the distance to the decision boundary. However, what *exactly* is going on in the high-dimensional space is difficult to visualise.

This problem also results in a very *adhoc* approach to improving classifiers. Choice of kernel, kernel parameters and penalty term C are done by looking at the *final* solution, and not the distribution of data in feature space. The solution does not always explain where the problem is, and whether or not another kernel or a different parameter set could produce a more uniform distribution.

[9] Similarly to d, σ is also a user-defineable *kernel-parameter*.

[10] In other words, feature space is the same as input space.

The use of kernels *has* proven to be effective, though. Practical results on numerous real-world data sets[11] have shown that the lack of knowledge about the input to feature space mapping during the training process is worth sacrificing.

2.7 Choosing the kernel and C

Which kernel is best? How can we select optimal values of the kernel parameters? These are obvious questions that arise with so many different mappings to choose from.

A more formal metric for choosing the best kernel is provided by the upper bound on the VC dimension ([40]). However, even though the VC dimension can be used to describe the complexity and flexibility of a kernel, it does not provide practical proof that the chosen kernel is "best". Unless the choice can be validated by numerous independent tests on different practical problems, methods such as cross-validation is still to be preferred when making kernel and kernel-parameter selections.

The penalty term, C, also needs to be defined by the user. Again, there is no easy method for selecting it's value aside from evaluating the resulting model's performance.

2.8 Bias in SVMs

Recall the training process of the linearly separable case (section 2.3.1). As mentioned, the pair $\{\mathbf{w}, b\}$ is found such that the decision boundary is placed exactly between the two hyperplanes H_+ and H_-. Also recall that the value of b can move the decision boundary in direction of $\mathbf{w}$ towards either of H_+ or H_-. This is why b is also denoted the *implicit bias*, since it's value is learned in the training process.

However, in some cases it is desired to have a decision boundary that is *not* learned in the training process and placed exactly between the two classes. For this purpose, an *explicit bias* can be introduced via the kernels. This is done by adding a term p to the kernel function. In the case of the polynomial kernel, this is typically done as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + p)^d$$

Adding the explicit bias leads to a different decision boundary. This can be used to push the boundary towards one of the two data sets. In most cases, this will not lead to improvements. However, it is often done in practical implementations to make the training process slightly more efficient and stable ([56]).

[11] The web page http://www.clopinet.com/isabelle/Projects/SVM/applist.html lists a few of the many results.

2.9 Practical Issues

To implement the SVM theory is not straightforward. Even though the quadratic optimisation problem in equation (2.30) can be solved using existing packages specifically designed to solve such a problem[12], these can typically only be used on small problems due to memory and run-time restrictions. The following two techniques are used to overcome this problem and make large-scale classification possible:

- *Chunking* ([88]). This technique splits the training set up into smaller subsets, where each subset is easily solvable by a quadratic programming package. The process can be described as follows: After the first initial subset has been optimised, the support vectors are stored, and the remaining training examples are thrown away. The next subset is then added to this *working set*, and optimisation is performed again. This process is repeated until all training data have been treated in the *working set*. Chunking speeds up training drastically. However, when solution is not sparse, the working set becomes large and the training process can become lengthy.

- *Sequential Minimal Optimisation* ([90]). In this optimisation procedure, the training problem is decomposed into tiny tasks of optimising only two of the a_is in equation (2.30). Solving such a problem can be done analytically and therefore does not need a quandratic programming solver.

These two techniques are implemented in two very popular toolkits used for the experiments[13]. However, even though they provide drastically reduced training times, the relation between amount of training data and training time still remains non-linear[14]. Consequently, large-scale classification is a time-consuming task.

Another important issue to bear in mind is *normalisation*. The toolkits requires the training samples to be normalised, otherwise the run-time will increase, and, in some cases, the optimisation process will fail to converge ([56]).

2.10 Multi-class SVMs

Many real-world data sets involve multiple classes, including the task of this thesis[15]. Since SVMs are inherently binary classifiers, techniques are needed to extend

[12] For example LOQO ([117]).
[13] SVM-Light [56] and SVMTorch [20]. Will be explained in chapter 3.
[14] Approximately $O(N^2)$.
[15] A class denotes a target label. “Classes” denotes the set of possible target labels. These terms will be used inter-changeably throughout the thesis.

the method to handle multiple classes. The goal of such a technique is to map the generalisation abilities of the binary classifiers to the multi-class domain.

In literature, numerous schemes have been proposed to solve this problem. For example [29], [77], [99], [39], [50], and [27]. This section will not try to describe all of these, but concentrate on two that have shown to produce good generalisation performance in practice. The choice has also been limited by the fact that the training time scales approximately $O(n^2)$ with the number of training samples n. Since the data set in this thesis contains over a million samples, multiclass schemes where binary classifiers contains all classes are not feasible. Among the dismissed schemes are: a) the One vs Rest scheme [50], b) Error Correcting Output Codes [22], and c) Binary Trees [7, 96].

2.10.1 One vs. One classifier

The One vs. One classifier is a system proposed by Friedmann ([29]), and has become the most popular and successful multi-class SVM method. The principle behind the method is very simple. It trains a binary model for each pair-wise combination of classes. In classification, each unseen example are evaluated on all binary models and classified to the class that "wins" most binary classifications[16]. This method is also called the *voting scheme*, since each binary classification assigns one "credit", or "vote", to the winning class.

To describe the classifier in a more formal way, let the set of classes be C. Furthermore, let the size of C be K. There will be one binary SVM for each class-pair $\{c_i, c_j\}, \forall c_i, c_j \in C \wedge i \neq j$. The resulting number of binary classifiers is $K(K-1)/2$, and each class is involed in $K-1$ models.

After training the $K(K-1)/2$ models, classification of unseen examples can be performed. Recall the function which classifies unseen examples (equation (2.34)). Let us denote the function associated with the binary classifier of $\{c_i, c_j\}$ as:

$$g(\mathbf{x})_{i,j} = sign(f(\mathbf{x})_{i,j}) \tag{2.38}$$

An unseen example, $\mathbf{x}$, is then classified as:

$$f_{1vs1}(\mathbf{x}) = \arg\max_i \sum_{i=1}^{K} \sum_{j=1 \wedge i \neq j}^{K} V_{i,j}(\mathbf{x}) \tag{2.39}$$

[16] A class "*wins*" a classification if the unseen example is classified to lie on it's side of the decision boundary in equation (2.34)

where:

$$V_{i,j}(\mathbf{x}) = \begin{cases} 1 & \text{if} g_{i,j}(\mathbf{x}) = 1 \\ 0 & \text{if} g_{i,j}(\mathbf{x}) = -1 \end{cases} \tag{2.40}$$

- The advantage of splitting up the multi-class problem into multiple binary sub-problems is that different decision boundaries can be created for each class-pair. This potentially results in a very complex decision boundary. Furthermore, due to the manner in which classification is carried out, even if an unseen example is misclassified by one binary SVM, it still has a chance of being correctly classified as each class is involved in $K - 1$ binary models.

- However, the classifier also has some obvious disadvantages. If the problem contains many classes, i.e. K is large, the number of binary SVMs required, $K(K-1)/2$, will consequently explode. This will also make classification slow as all classifiers need to be evaluated before a decision is made.

  In addition, there is another flaw in the scheme: each binary classifier is only trained to choose between two classes. Therefore many votes are *forced* to be cast for incorrect classes. In practice, these votes are distributed sufficiently uniformly as to not overrule votes from those binary classifiers who are choosing between a pair of classes which includes the correct class.

Nevertheless, this classifier has produced some of the best results on multi-class tasks found in SVM literature ([77]).

2.10.2 DAGSVM

DAGSVM stands for *Directed Acyclic Graph SVM.* It is a multi-class method proposed by Platt ([53]), which employs the exact same training phase as the One vs. One classifier. I.e. it creates $K(K-1)/2$ binary classifiers. However, it distinguishes itself in the classification phase by constructing a rooted binary tree structure with $K(K-1)/2$ internal nodes and k leaves (see figure 2.6, below). Each node in the graph contains a binary classifier of ith and jth classes.

Classification is performed as shown on the figure.

- The advantage of using the DAGSVM is that it only needs $K - 1$ evaluations to classify an unseen example. Appreciably, we still retain the complex decision surface from the One vs. One classifier.

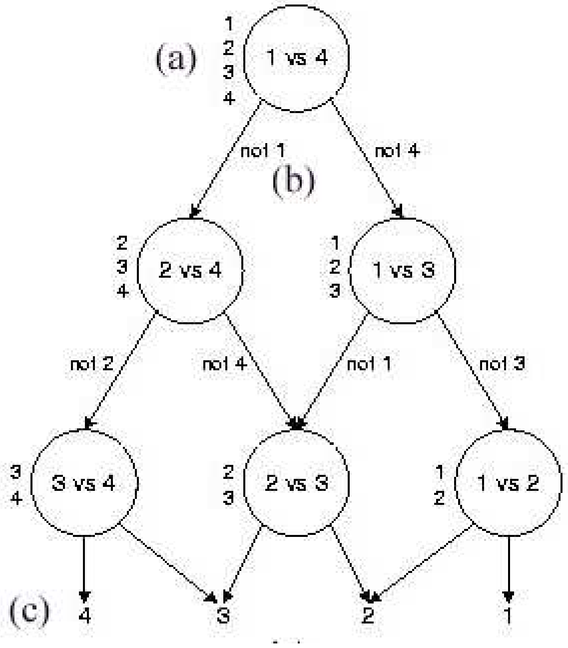

Figure 2.6: Decision graph for a classification task with 4 target classes. Starting at the root node (see (a) on the figure), the SVM belonging to the node is evaluated, using equation (2.12). (b) It moves down the tree to either left or right child depending on the outcome. (c) It repeats this process until a leaf k is reached, and assigns the unseen example to the k th class (from [53]).

- However, it is not necessary superior to the One vs One scheme. A disadvantage is it's lack of stability: In classification, if *one* binary misclassification happens, the unseen example will be misclassified. As mentioned earlier, this is not always the case with the One vs. One classifier, and thus the robustness of this method may not be as high.

2.11 Conclusion

It is not obvious which of the multi-class methods are best for extending the generalisation powers of the binary SVMs. Since no theoretical metrics are available to assess their generalisation abilities, it is not possible to make proper comparisons without evaluating their capabilities on real-world data sets.

When constructing and evaluating a multi-class method, it is easy to forget the importance of the underlying binary SVMs. This must be not be done given that the generalisation ability of the multi-class classifier is fully dependent on the generalisation abilities of the binary SVMs. An analysis of the individual binary SVMs should be carried out before the extension to the multi-class case is done.

This ends the introduction to Support Vector Machines. There are numerous other aspects that could be considered, though. For example, describing SVM in the context of the wider family of linear discriminant classifiers could give a better picture of it's placement amongst other learning machines[17]. Furthermore, a more detailed theoretical analysis of the generalisation bounds of binary SVMs could lead to a better understanding of their generalisation performance (see [4]).

However, it is hoped that this chapter have covered enough aspects to give an understanding of the issues involved in working with SVMs and employing the method on a practical problem.

[17] Refer to [42] for details.

Chapter 3

Experiments

In this chapter, I will evaluate the performance of SVMs on the problem of *frame-by-frame* classification. The problem is not easy. To give an indication of just *how* difficult it is, I have plotted the two first dimensions of frames from two target phoneme classes in the data set. The result is shown on figure 3.1. As can be seen, the plot displays an extreme amount of overlap. A potential learning machine must somehow attempt to separate such classes and produce valid predictions.

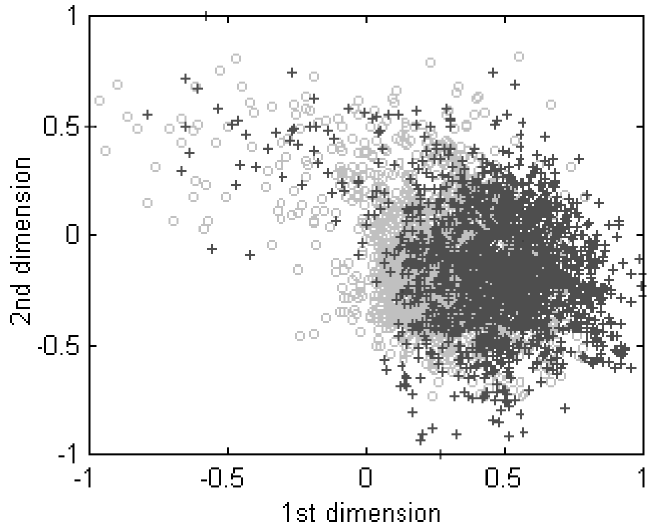

Figure 3.1: Plot of the first two dimensions from /aa/ and /ae/ vowel speech frames.

In the following sections, results from three groups of experiments are reported. First, analysis of models trained on class-pairs is performed to learn how to get the best generalisation performance out of each individual SVM. Subsequently, two groups of multi-class experiments are explained. These are related in the sense that they each utilize one of the multi-class classification methods described in section 2.10.

Note that, since the work of this thesis is a continuation of the work done in [96], some of the experiments described below were also reported in [96]. The descriptions are given here to give a better understanding of the progress from simple preliminary experiments to the extensive multi-class experiments. In other words, it is done to avoid jumping directly to the more extensive experiments without first explaining the conclusions and examinations on which they are based.

3.0.1 Toolkits used

Throughout the experiments, I used the toolkit SVM-*Light* ([56]). It implements the *Chunking* algorithm described in section 2.9. Even though it is not the fastest toolkit available, it returns a lot of useful information about the training problem. This includes estimates of the generalisation error and VC dimensions. For larger problems, when more than 16000 examples were included, the SVMTorch toolkit was used ([19]). This is faster due to the implementation of the *Sequential Minimal Optimisation* algorithm also described in section 2.9.

3.0.2 The TIMIT speech database

The data set used for the task is the TIMIT database[1] ([34]). It is a corpus of high-quality continuous speech from North American speakers, with the entire corpus reliably transcribed at the word and surface phonetic levels. The speech is parameterised as 12 Mel-frequency coefficients (MFCC) plus energy for 25ms frames, with a 10 ms frame shift[2]. The target labels consists of 40 different classes, representing 40 phonemes from the English language. This is a configuration commonly used in speech recognition ([92]).

The corpus is divided into 3648 training utterances and 1344 test utterances (only si and sx sentences were used). No speakers from the training set appear in the test set, making it 100% speaker-independent. To measure performance, a validation set of 1000 utterances was subtracted from the training set. For generating training and test samples from the three data sets I randomised the order in which the utterances were chosen[3].

In appendix B, an overview of the TIMIT database is shown.

[1] The description of the TIMIT database is based on [61].

[2] Note that delta and acceleration coefficients (single and double derivatives) can be derived from these 13 coefficients, resulting in a total of 39 coefficients.

[3] This was only done *once* to allow direct comparison between experiments.

3.1 Binary Experiments

Each binary SVM involves a separate optimisation problem. As mentioned before, in a multi-class problem it is of great importance that each of these generalise as well as possible. This is the case, no matter if it is a simple linearly separable problem, or requires a complex decision boundary.

Several user-defined parameters are involved when training an SVM model. These need to be learned such that the discriminate performance of each SVM is optimal. The parameters are summarised in table 3.1. Additionally, the important factors in the resulting models are listed in table 3.2. How these are related to the user-defined parameters will be examined in this section.

Kernel	A kernel function needs to be chosen. The choices include linear and non-linear kernels, for example the ones described in section **??**. The kernels often involves one or more kernel specific parameters to be set. Natutally, these need to be optimised.
Penalty term C	This parameter regulates the relation between the minimisation of $\|\|\mathbf{w}\|\|^2$and minimising the classification error (in equation (2.22)).
Composition of feature vector	The feature vectors of the input data should naturally contain as much information about the under-lying problem as possible.
Amount of training data	Generally, performance increases when more training data is included. Unfortunately, as mentioned in section 2.9, the relation between training time and size of training set is non-linear. A trade-off between generalisation performance and run-time must be made.

Table 3.1: Overview of user-defined parameters in a binary SVM

Number of support vectors (SVs)	The number of SVs in the resulting model is implicitly chosen by the user through the parameters listed in table 3.2. How are these related to the other factors should be investigated?
Training time	How the training time of the SVMs relate to the amount of training data used should be examined.
Classification time	Classification time is important if the classifier is to be used in any system where run-time is critical.
Performance on validation set	The most important issue of this thesis is to find the best possible performance. How this can be achieved should naturally be explored.

Table 3.2: Important factors in an SVM model

3.1.1 Experimental setup

By default, each feature vector contains 13 dimensions as described above. I created training sets of binary class-pairs, each consisting of data from two of the 40 phoneme classes with labels "+1" and "-1" as described in section 2.2.1.

3.1.2 Choice of kernel and C

To examine the effect of different kernels, I performed experiments using the three kernels from section 2.5. parameters were varied. The results are shown on figure 3.2[4]. The plots are found by a SVM model separating data from phoneme classes /aa/ and /ae/. Similar behaviour was found in all examined SVMs[5].

Looking at the figures it can be concluded that a search for optimal parameters is needed. This is easily done for the linear kernel, but not as straightforward to do for the non-linear kernels. The kernel specific parameter is strongly related to the penalty term C, and hence, it is not possible to search for the parameters individually. Fortunately, the areas around the peaks are reasonably smooth. This makes it possible for even a rough search is likely to find near-optimal parameters.

Nevertheless, techniques for automatically learning the SVM parameters were investigated. Several such schemes exist in literature ([59], [69], and [121]). I examined some of these, but unfortunately, either the increase in performance by employing these methods were marginal (mostly less than 0.1%) or the required run-time far exceeded the run-time of the exhaustive search.

3.1.3 Including different amounts of training data

In this experiment, I investigated how the size of the training set affected: a) the training time, b) the number of support vectors in the model, and c) the classification time. Since the training time increases near-quadratically with the size of the training set, the number of training examples must be restricted to reduce run-time. This is particularly important, considering the number of binary SVMs required in the multi-class classifiers.

On figure 3.3, three plots shows the effect of varying the amount of training data[6].

Based on the above experiments, the following observation can be made:

[4] On figure C, *Gamma* is used instead of σ as kernel parameter. The relation are: $Gamma = 1/2\sigma^2$. The SVM-*Light* toolkit uses *Gamma*, and thus this factor will be used instead of σ for the remainder of the thesis.

[5] In total, around 20 different class-pairs were examined.

[6] Similarly to the above experiment, the plots are found by a SVM model separating data from phoneme classes /aa/ and /ae/ using the Gaussian kernel. Comparable plots were found for all examined SVMs.

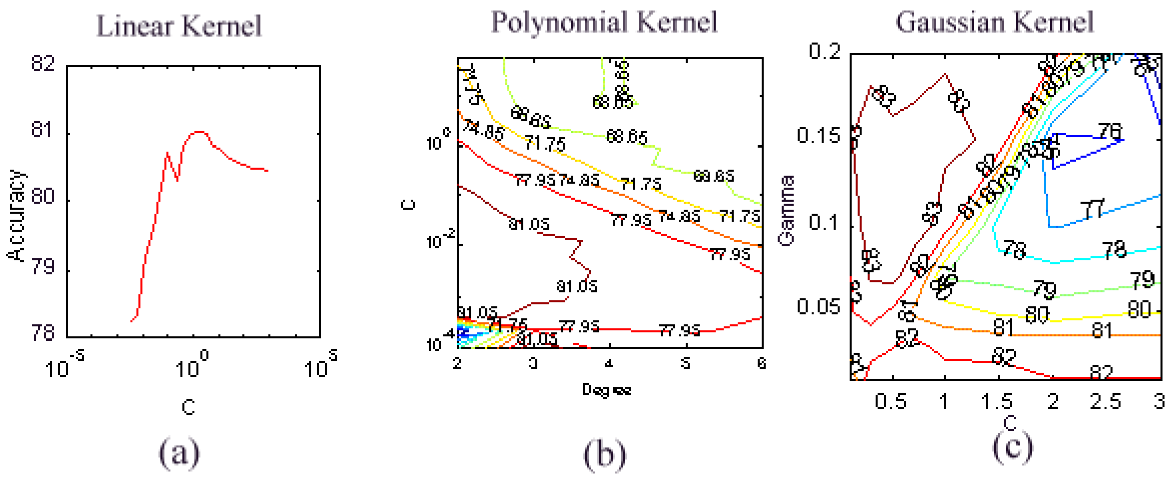

Figure 3.2: Effect of C and kernel parameter on validation accuracy. Plot (a) shows the resulting performance of the linear kernel when varying C. Plot (b) displays performance of the polynomial kernel when varying C and degree d . Finally, plot (c) shows performance of the Gaussian kernel when varying C and Gamma. Notice how the Gaussian kernel shows the best performance, although its decision surface is less smooth than the surface of the other kernels.

- The Gaussian kernel seems to offer the best generalisation performance. This was expected: the linear and polynomial uses a feature space with fixed number of dimensions. In contrast, the Gaussian kernel has the potential to map the data into a space with *infinite* dimensions, which intuitively gives it greater flexibility. However, it is not always superior. In a few cases, the polynomial kernel yielded comparable performance.

- For this problem, near-optimal parameter sets can be found using a rough exhaustive search through parameter space due to the smooth maxima of the decision surfaces. This applies to all examined kernels.

- The problems of *scaling* the SVM to handle large amounts of training data becomes apparent when looking at figure 3.3(b). The near-quadratic increase in training time severely restricts the amount of training data that can be used. In comparison, Recurrent Neural Networks have a clear advantages over SVMs. They do not have this problem as training time simply scales linearly with the size of the training set.

- By increasing the amount of training data, the number of support vectors is also increased. This prolongs classification time as the contribution of each individual support vector needs to be computed in equation (2.33).

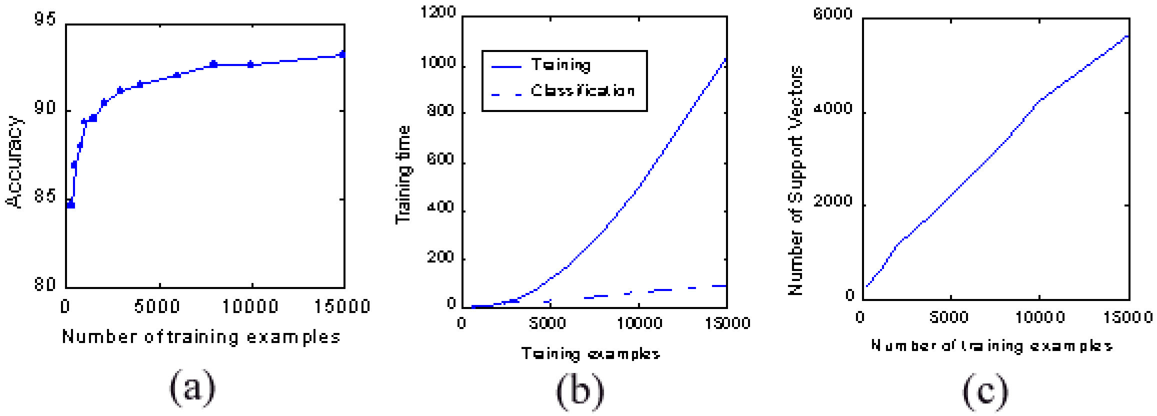

Figure 3.3: The effect of varying the training set size. Plot (a) shows the increase in generalisation performance when varying the size of the training set. As expected, the performance increases when the amount of training data increases. Plot (b) show that the relation between training set size and training time is near quadratic. Fortunately, classification time only increases linearly with increase in training set size. Finally, plot (c) displays the roughly linear increase in number of support vectors when increasing the amount of training data.

All these conclusions will be taken into considerations when the multi-class experiments are performed.

One user-defined parameter has not been discussed above: the composition of the feature vector. Experiments were carried out using various feature vectors. However, these will not be described in this section, since the benefits of using the different feature vectors did not become apparent until they were used in multi-class problems.

3.2 Selecting multi-class methods

In section 2.10, I described two different methods for extending the binary SVM to handle multi-class data: the One vs. One and DAGSVM classifiers. In the following sections, I will start out describing the One vs. One classification experiments in details. The other classification scheme will be explained much more briefly, since it can take advantage of the conclusions drawn from the group of One vs. One experiments.

3.3 One vs. One Classifier

3.3.1 Introduction

In this section, four sets of experiments will be described based on the One vs One classification scheme. These will examine the effects the different parameter choices have on performance.

Recall the description of the One vs. One classifier in section 2.10.1: this classifier creates one binary SVM model for each combination of classes, resulting in $K(K-1)/2$ SVMs. When an unseen example is classified, all models are evaluated. For each classification, a "vote" is given to the "winning" class[7]. The unseen example is classified as the class with most votes.

This method is one of the most popular and successful multi-class schemes found in literature ([11, 10]). In numerous cases, including frame classification, this classifier has shown excellent generalisation performance ([17, 15]).

The task in focus is frame classification. It has 40 possible target labels, where each target corresponds to a phoneme class. This results in $\frac{40\cdot(40-1)}{2} = 780$ binary classifiers. This is a considerable number. Each one needs to be constructed such that it achieves optimal generalisation performance.

3.3.2 Practical issues

As one can easily imagine, training and evaluating 780 binary SVM is a very comprehensive task. Even when using small amounts of training data, the training process can take days. Moreover, all 410920 examples contained in the TIMIT test set[8] must be evaluated on the 780 binary SVMs. This results in over 300 million evaluations, and would take weeks to complete on "normal" machines (Intel Pentium PCs or Sun Ultra-5s). For this reason, I have reduced the amount of test examples to 4000 per class in the first experiments [9]. This should still be enough to give a reasonably accurate measure of performance. Moreover, by means of the frequencies of the individual phonemes in the test set, it is possible to give a fairly precise estimate of the full test set accuracy.

3.3.3 Experiments 1: Effects of kernels

This experiment is done to get an indication of the performance of the One vs. One classifier system and, at the same time, compare the generalisation abilities of the different kernels.

[7] As mentioned earlier, a class *"wins"* a classification if the unseen example has been classified to lie on the classes' side of the decision boundary in equation (2.34)

[8] See Appendix B for an overview of the phonemes in the TIMIT test set

[9] Not all classes have 4000 examples in the full test set (see Appendix B). Consequently, these class sets were not reduced.

Experimental setup

To begin with, I chose to include only 1000 training examples per class, resulting in 2000 training examples per classifier. I used the 13 basic MFCC coefficients described in section 3.0.2, and varied the kernel and C parameters through a rough exhaustive search. The created models were tested using 4000 test examples per class.

I took advantage of having several Sun UNIX machines available (Ultra-5 and SunFire). To reduce run-time, I split up the big job into numerous sub jobs, and distributed these to different machines. While the experiment previously would need weeks to finish, this reduced the experiment's total run-time to approximately 70 hours.

Kernel	**MBA Val**	**MBA Test**	**$\mathbf{MA}_{4000}$**	**$\mathbf{MA}_{estfull}$**	**Top 3 / Top 5**
Linear	89.8	82.0	27.3	33.9	54.6/65.6
Polynomial	90.9	83.4	29.4	34.4	56.0/68.1
Gaussian	**91.2**	**85.4**	**46.0**	**50.9**	**62.1/85.2**

Table 3.3: Performance of the One vs. One classifier using different kernels. From the multi-class accuracies reported (MA_{4000} and $\mathrm{MA}_{estfull}$), it is clear that the Gaussian kernel outperforms both the linear and polynomial kernels. The best result is 46.0%/50.9%, and best Top 3/5 is 62.1%/85.2%.

Results and conclusions

The result of the experiments are shown in table 3.3[10]. Looking at the results, the performance of the kernels seem to confirm the conclusion made in section 3.1.2: the Gaussian kernel offers superior generalisation due to it's greater flexibility. However, the best-found result of 46.0%/50.9% is not impressive considering the extensive run-time. Moreover, compared with the best results found in literature, they are rather poor (70.4% in [94] and 74.8% in [13]). The explanation can perhaps be found in the binary SVM's performances on the validation and test sets. Notice the difference between MBA Val and MBA Test. Apparently, the validation set and the test sets are

[10]Note on notation used in table. MBA stands for *Mean Binary Accuracy*. It is the average of all accuracies found from evaluating the test or validation set on the 780 binary SVMs. "MBA Val" is the MBA of the validaton set, and "MBA Test" the MBA on the test set.
MA_{4000} and $\mathrm{MA}_{estfull}$ denote *Multi-class Accuracies*. MA_{4000} is found by evaluating the classifier on 4000 test examples per class. $\mathrm{MA}_{estfull}$ is found by weighting the results from MA_{4000} with the phoneme frequencies in the full TIMIT test set. Intuitively, $\mathrm{MA}_{estfull}$ should offer a good estimate of the performance on the complete test set.
Top 3/Top 5 are alternative metrics for measuring classifier performance. "Top N" denotes the percentage of the test examples where the correct class is among the set of N phonemes with most votes. E.g. when Top 5 is 65.63%, it means that in 65.63% of the test examples the correct phoneme frames were amongst the 5 phonemes with most votes.
These terms will be used in the remainder of the thesis.

somehow very different. The reason for this can be found in the composition of the feature vectors. They contain only the basic 13 MFCC coefficients described in section 3.0.2. As observed in [85], the 13 coefficients used have large within-class variances. Thus the risk of large differences between validation and test sets is high, and leads to poor performance.

3.3.4 Experiments 2: Adding more features

The problem of large variances in the data sets may be alleviated by adding first and second order derivatives to the 13 existing MFCC coefficients. These can be computed over several frames to comprise the dynamics of the speech and help discriminating between fast and slow changing phonemes. E.g. generally vowels are very slowly changing whereas ficatives are not. These derivatives are commonly used in speech recognition, and are typically found to improve classification results ([92]).

Experimental setup

After adding the new features, each feature vectors contained the 13 existing MFCC coefficients, 13 first order derivatives and 13 second order derivatives, i.e. 39 features in total. I chose to run the experiment using the Gaussian kernel since it had shown the best generalisation performance. The result is shown in table 3.4, where it is compared with the previous result.

Features	**MBA Val**	**MBA Test**	**MA$_{4000}$**	**MA$_{estfull}$**	**Top 3 / Top 5**
13	91.2	85.4	46.0	50.9	62.1/85.2
39	**95.1**	**89.7**	**51.1**	**53.6**	**68.7/87.5**

Table 3.4: Effect of adding derivatives. The generalisation performance is increased in both overall and binary accuracies. In particular, notice the substantial increase in performance in binary accuracy.

Results and conclusions

As can be seen from the table, the overall performance is increased by 2.7%. Moreover, notice how the binary accuracies of the validation and test sets have increased by almost 4%. In other words, the classification error of each individual SVM have decreased by 46%. This significant improvement confirms that: a) the derivatives contain important information about each frame, and b) even with the increased number of dimensions, the SVMs are capable of detecting the features that characterises the data. This demonstrates the theoretical claim that SVMs counteracts the *curse of dimensionality* (from section **??**).

Furthermore, since the first and second order derivatives are computed over several frames, the improvement indicates that a substantial part of a frame's information lies in the context frames. This conclusion was also reached by [32] and in the extensive examinations of [74] and [43]. Investigations into adding additional context information seem reasonable.

3.3.5 Experiments 3: Adding context frames

How can we include more contextual information about each frame? The answer lies in the neighbouring frames. Speech is fairly slowly changing, and each phoneme lasts on average 65ms [43]. Given that speech frames are extracted every 10ms, this results in 6.5 frames per phoneme on average. Consequently, contextual information can be added by including neighbouring frames in the feature vectors.

Experimental setup

Several ways were explored of adding neighbouring frames, or *context frames,* to the feature vector. Each feature vector included the frame to be classified (denoted **T** as in *target frame*) and various context frames $\mathbf{C}_i$ on each side:

$$\mathbf{x} \equiv \langle \mathbf{C}_{-N}, \ldots, \mathbf{C}_{-1}, \mathbf{T}, \mathbf{C}_1, \ldots, \mathbf{C}_M \rangle$$

where $\mathbf{C}_{-i}$ denotes the context frame positioned i frames *before* the target frame **T**, and similarly, $\mathbf{C}_i$ represent the i th context frame *after* the target frame.

Each frame used the 39 features per frame described in the previous section, since they were shown to outperform the basic 13 features. To be able to compare with earlier results, the Gaussian kernel and 1000 training examples from each class were used.

Feature vector $\mathbf{x}$	$\lvert\mathbf{x}\rvert$	MBA Val	MBA Test	MA_{4000}	$\mathrm{MA}_{estfull}$	Top 3/Top 5
$\langle \mathbf{T} \rangle$	39	95.1	89.7	51.1	53.2	68.7/87.5
$\langle \mathbf{C}_{-1}, \mathbf{T}, \mathbf{C}_1 \rangle$	117	95.5	94.8	58.9	60.8	82.4/90.1
$\langle \mathbf{C}_{-2}, \mathbf{C}_{-1}, \mathbf{T}, \mathbf{C}_1 \rangle$	156	**95.6**	94.9	59.5	62.9	83.1/90.7
$\langle \mathbf{C}_{-2}, \mathbf{C}_{-1}, \mathbf{T}, \mathbf{C}_1, \mathbf{C}_2 \rangle$	195	**95.6**	**95.0**	**59.8**	**63.7**	83.3/**91.0**
$\langle \mathbf{C}_{-3}, \mathbf{C}_{-2}, \mathbf{C}_{-1}, \mathbf{T}, \mathbf{C}_1 \rangle$	195	95.3	94.7	58.9	63.1	**83.3**/90.9
$\langle \mathbf{C}_{-3}, \mathbf{C}_{-2}, \mathbf{C}_{-1}, \mathbf{T}, \mathbf{C}_1, \mathbf{C}_2 \rangle$	234	95.2	94.7	58.8	62.9	83.4/90.7
$\langle \mathbf{C}_{-3}, \mathbf{C}_{-2}, \mathbf{C}_{-1}, \mathbf{T}, \mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3 \rangle$	273	95.2	94.6	58.6	62.5	82.7/90.4

Table 3.5: Summary of experiments with different feature vectors. The results show that including context frames substantially improves performance. The best configuration is found when two context frames are included on each side of the target frame. This improves performance by a whole 8.7%/10.5% from previous found results.

Results and conclusion

Results from five different feature vector compositions are shown in table 3.5. As seen on the table, including context frames substantially improves performance. The explanation for this improvement can be found in the much-improved MBA Test. The difference between MBA Val and MBA Test has virtually disappeared, which indicates that the in-class variance in the data has decreased. Therefore, it can be concluded that information "hidden" in the context frames is vital for generalisation performance.

The best result found in literature, 74.2% by [13], may still seem substantially better than the 59.8%/63.7% found in this experiment. However, the result should not be dismissed so easily. [13] also report a 60.9% frame accuracy using only 10% of the training data. Since my experiments are based on approximately *half* of this amount of training data, I believe them to be very promising.

3.3.6 Experiments 4: Including additional training data

In the previous experiments, each binary model has included only 1000 training examples per phoneme class, resulting in 2000 training examples per classifier. This has been done to restrict the run-time of the experiments. However, recall figure 3.3(a). The figure indicates that the performance of the binary classifiers could improve if additional training data is added. Consequently, this could lead to better generalisation performance.

Experimental setup

I ran three experiments with different amounts of training data. The first experiment contained 4000 training examples per class, and the second one a staggering 16000 examples per class. This is, respectively, 14% and 40% of all training samples[11]. Since training the SVMs would take a substantial amount of time, it was not feasible to perform parameter search. Instead, I reused the "best" parameters found in section 3.3.5. Although this was not the optimal solution, it was required to reduce run-time. Moreover, the job was split up into numerous sub jobs and run in parallel.

The final experiment involved evaluating the first 16000-example experiment on the *full* TIMIT test set.

[11] Far from all classes had 16000 samples in the full training set (see Appendix B). Consequently, all samples from these classes were used.

Examples per class	MBA Val	MBA Test	$\mathrm{MA}_{4000}/\mathrm{MA}_{estfull}$	Top 3/Top 5
1000	95.5	95.1	**59.9**/63.8	83.3/91.0
4000	96.1	95.6	56.7/67.3	84.7/92.5
16000	96.5	**96.1**	55.9/**70.8**	**86.3/93.5**
16000 (full)	96.5	**96.1**	70.6	86.1/93.4

Table 3.6: Results of experiments with additional training data. As expected, the 16000-example experiment shows the best estimated accuracy. This was confirmed by testing the model on full TIMIT test set. The result of 70.6% accuracy is shown on the last row.

Results and conclusions

The results from the experiments are shown in table 3.6, where they are compared with the previous best result. Two conclusions can be made from looking at the table. First of all, the estimated and actual full test set accuracies are comparable. This confirms the accurateness of the previously computed estimates. Secondly, the 70.6% accuracy found on the full test set is better than the second best result found in literature (70.4% by [94]). This proves that SVMs is a viable and capable technique for framewise phoneme classification.

Another important observation can be made. Notice how the unadjusted overall performance (MA_{4000}) *decreases* when the amount of training data increases. Although this may come as a surprise, a logical explanation exists:

As mentioned above and seen in Appendix B, some phonemes are very infrequent and have few training samples in the training set. Consequently, this results in an uneven distribution of examples during training. Recall the objective function (equation (2.22)):

$$\|w\|^2 + C\left(\sum_i \xi_i\right)^k$$

The SVM training algorithm aims to minimise this expression, which partly means minimising the classification errors (the second term). As described in section 2.3, each ξ_i is defined as the distance from a misclassified training example to the decision boundary. Hence, if one set of training examples is much larger than the other, it will also have the majority of ξ_is. In effect, to minimise the second term, the hyperplane will be moved in the direction of the smaller class.

This creates the implicit bias towards larger phoneme classes, and the larger classes will "win" more classifications. This explains the improved classification performance on the full test set. However, as a result of this bias, the smaller classes will "lose" more classifications[12], and the unadjusted classification rate will decrease.

[12] As opposed to "winning" classifications.

A final note should be made about the *sparsity* of the solution, i.e. how many support vectors are contained in the solution. Each model binary classifier used on average 2993.3 support vectors, which is about 26.9% of the training samples. A high number of support vectors (and therefore more dense solutions) are mostly found in classes with high overlap, because the model includes all samples that lies on the wrong side of the decision boundary (i.e. if $\xi_i \neq 0$ as mentioned in section 2.3.2).

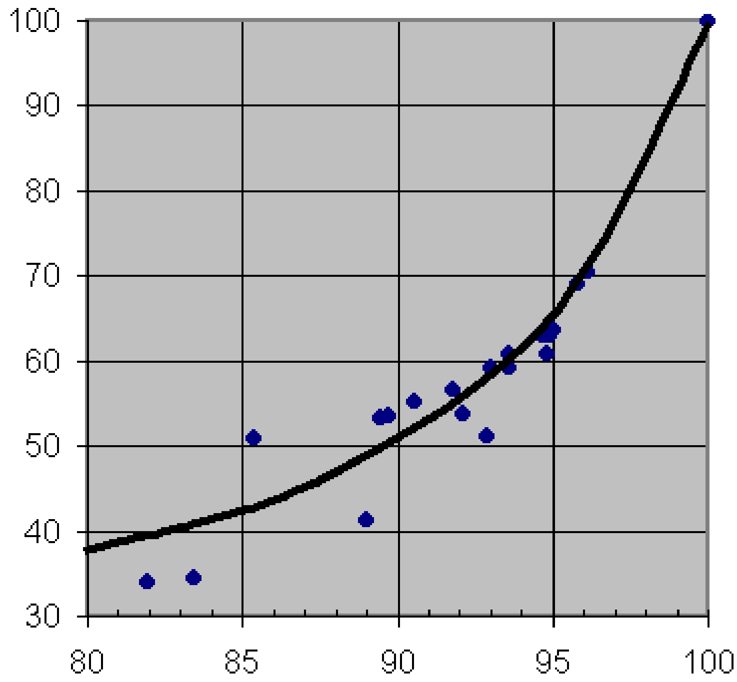

Figure 3.4: Binary mean accuracy (x axis) vs. overall accuracy (y axis) using the voting scheme.

3.3.7 Generalisation performance

To measure the generalisation performance of the One vs. One multi-class scheme, consider how the overall accuracy is related to the binary accuracy of the 780 binary models. It seems logical that if *all* 780 binary classifications had a 100% test accuracy, this would lead to a 100% overall accuracy.

For this reason, I collected all results from the above experiments, and plotted the relationship between the two factors. The relationship is displayed on figure **??**.

The plot shows a rather positive fact: looking at the estimated trend line, even a slight increase in binary accuracy should lead to a substantial increase in overall accuracy.

3.3.8 Conclusion

In the above experiments we have seen that the One vs. One classifier offers good generalisation ability. The completed experiments were by no means performed under optimal conditions, and still the classifier performed extremely well. The best-found result of a 70.6% frame classification rate is competitive with the second best result reported in literature (70.4% in [94]). Interestingly, the result was achieved using only 40% of the training data.

Conclusions can not only be made about the multi-class One vs. One scheme, but about the SVM method in general. An average binary accuracy of 96% shows that SVMs are a capable discriminant classifier that can generalise well.

However, one may be concerned about the extensive number of binary models required to build the multi-class model. The 2993 support vectors needed per model results in a total of just under 2 million components[13], and classifying an unseen example involves evaluation of all 2 million components. Even though run-time is not the focus of this thesis, the classifier may seem overly cumbersome to ever be practical in a real-time speech recognition system. However, it is easy to improve classification speed. Currently, the classifier evaluates *all* 780 binary SVMs before it makes the prediction. If some of the classes could be dismissed earlier, the number of evaluations could be reduced dramatically. From the experiments, I observed that phonologically similar phonemes tended to produce similar classification results on different binary SVMs. For example, a vowel phoneme "winning" a binary evaluation against a consonant phoneme indicates that the unseen phoneme might be a vowel. This can be exploited in classification to *prune* the number of evaluations. As a result, evaluating a few chosen SVMs (less than 10) gives a fairly precise indication of what phonological group an unseen example belongs to. The resulting number of evaluations is suddenly only a fraction of the 2 million evaluations.

3.4 DAGSVM classifier

3.4.1 Introduction

Platt's DAGSVM algorithm [53] was described in section 2.10.2. To briefly summarise, it uses the same training phase as the One vs. One classifier but distinguishes itself in the classification phase by constructing a rooted binary tree structure with $K(K-1)/2$ internal nodes and k leaves. Each node in the graph contains a binary classifier, and classification of an unseen sample is done, starting at the root node, by moving down

[13] Total number of support vectors multiplied by number of features in the support vectors. This figure is about 26.9% of the total training data.

the tree following the left or right branch of a node depending on the outcome of each node's classification. The leaf it reaches indicates the classification of the unseen samples.

In [50] and [53] the DAGSVM algorithm were shown to perform similarly and in some cases *better* than the voting scheme. It is hoped that this performance can be reproduced in this section.

Implementation issues

[53] describes an easy way to implement of the algorithm. We can represent the tree as a list of possible target labels (the phoneme classes). The classification of an unseen example can then be composed of the following steps:

1. Evaluate the SVM for the two outer-most phoneme classes in the list.
2. The phoneme class which "loses" the classification is removed from the list.
3. Step 1 and 2 are repeated until the list contains only one phoneme class.
4. The unseen example is classified as the one remaining phoneme class.

[53] also discusses the composition of target labels in the list. He observes that the order in which phoneme classes are put in the list does not have any affect on performance. However, in this section, I have run experiments with both random and sorted lists[14].

Experimental setup

Since the classifier utilises the exact same training phase as the One vs. One classifier, it could simply reuse the best models from the One vs. One experiments (section 3.3.6). The algorithm was tested on the full TIMIT test set.

Composition of list	MBA Test	Accuracy
Unsorted	96.1	70.5%
Sorted	96.1	71.4%

Table 3.7: Classification results using the DAGSVM algorithm. The result of 71.4% improves upon the previous best result. Moreover, notice the accuracy improvement when sorting the lists.

[14] In the sorted list, the phonemes were sorted by their phonological groupings. I.e. vowels/semi-vowels, nasal/flaps, stops, and fricatives (see [16] for details). Evaluations could be skipped if phonemes appeared to be from another group.

3.4.2 Results and conclusions

The results can be found in table 3.7[15]. As it can be seen from the table, the DAGSVM classifier shows comparable generalisation performance to the One vs. One classifier. This is consistent with results reported in literature on other classification problems ([53] and [50]). Appreciably, as a result of sorting the list the result is slightly better than the previously best result (from section 2.10.2). This reaffirms the SVM method's capability as a phoneme classifier.

3.5 Conclusion

The above experiments have proven that SVMs can create capable and well-performing learning machines for frame by frame phoneme classification. Even with the tremendous amount of overlap amongst classes, it could successfully model the speech data and produce liable classifications. Moreover, through maximising the margin of separability between classes and thereby implementing the Structural Risk Minimisation principle, it demonstrated that it could counteract the *curse of dimensionality* and generalise well even when the data contained numerous irrelevant features. The best result found, 71.4% by using DAGSVM in section 3.4, is competitive with the best results reported in literature. Although it does not surpass *the* best result in literature of 74.2% reported in [13], it is slightly better than the 70.4% reported in [94]. This is noteworthy given the latter has been in development for years, and that it only uses 40% of the training data.

As mentioned earlier, the experiments in this chapter were by no means performed under optimal conditions. The three major difficulties faced during the experiments were: a) the number of binary SVMs needed in the multi-class schemes, b) the bias in the binary classifiers, and c) training time.

The amount of binary model needed in the multi-class schemes made the experimental process very extensive. Training and testing 780 models required a lot of planning, and many practical problems arose from simply trying to control and verify that the experiments were running properly. Combined with the long time required to train each binary classifier, the run-time of each experiment seriously affected the progress rate.

Regarding the bias, it was observed that the built-in implicit bias in the large-scale binary classifiers increased overall accuracy. However, there was not control over the inter-dependencies amongst the models. Recall from section 2.8 that it is possible to explicitly incorporate a bias into the SVMs. By adjusting the biases, a more structured

[15] In the table, "MBA Test" denotes the average accuracy for all evaluated nodes.

method could be introduced to directly optimise the overall generalisation performance of the multi-class classifiers.

Finally, the amount of samples used in training severely affected the training time. The SVM's training time increases near-quadratically with the amount of data, so only 40% of the total TIMIT training data was used to train the models. Since more training data was shown to increase accuracy, a key to better generalisation performance is to solve the problem of how training time scales with the training set size.

However, a solution to the scaling problem is not straightforward. Not only should the potential method be able to promptly handle large amounts of training data, but it also needs to consider memory requirements and naturally, produce optimal solutions. Does such a method exist? Part II of this thesis will attempt to answer that question.

Chapter 4

Probability outputs

Before searching for an improved method, one vital improvement to the SVM method can be made. Almost all speech recognition systems are based on statistical models ([58]). If the multi-class systems from the previous chapter should be used in a complete speech recognition system, a probabilistic interpretation of the classification results is needed. Recurrent Neural Networks (RNNs) have a clear advantage over SVMs on this point: the outputs of RNNs can be interpreted as posterior probabilities ([9]). SVMs, on the other hand, have no clear way of interpreting outputs as probabilities. Via equation (2.33), they return the classified instance's distance to the decision boundary. From conclusions found in [32] and [91] there is no clear relationship between this distance and the posterior class probabilities. However, if we can find such a relationship, it will be possible to generate probabilities from the class predictions.

4.1 Estimating posterior probabilities

One scheme for producing posterior probabilities from the One vs. One multi-class SVM classifier is very simple: the number of votes accumulated by each class roughly correspond to a confidence measure in the class. Classes with a lot of votes have won a lot of binary classifications, and thus logically seem more likely to be correct. A very rough estimate of posterior probability could be obtained by scaling down the number of votes by $1/780$ such that the sum of all votes are 1. These values could be interpreted as probability estimates.

However, this scheme is unlikely to work very well, and even if it did, would only work for the One vs. One classifier. To obtain a more accurate estimate the individual SVMs must be considered.

One approach is to fit a Gaussian to the class conditional probabilities $P(f\,|y=1)$ and $P(f\,|y=-1)$. The resulting posterior probability $P(y = 1\,|\,f)$ will then be a sigmoid [119]. Unfortunately, the Gaussian assumption needed to use the class-conditional probabilities are typically not satisfied in SVMs because of the asymmetric distributions of the output distances (see figure 4.1). [91] proposes to fit the posterior probabilities directly onto the SVM. Since the posterior probability is a sigmoid, a sigmoid function can be fit to the classification outputs. The posterior probabilities can then be estimated by:

$$P(y = 1|f) = \frac{1}{1 + e^{Af+B}} \tag{4.1}$$

The sigmoid is fit to the model by estimating the two parameters A and B . This can be done by maximum likelihood estimation. [91] discusses a potential problem of fitting both sigmoid and SVM parameters from the same data set. In non-linear models, the support vectors typically correspond to a large subset of all training data in non-linear models, and because the support vectors defines the decision boundary, the resulting sigmoid will be biased towards the decision boundary (i.e. the sigmoid will be "steeper" than it should be). One of the possible solutions proposed in [91] is to use a separate validation set. This is easy in this case, since a validation set is already used to learn the SVM parameters.

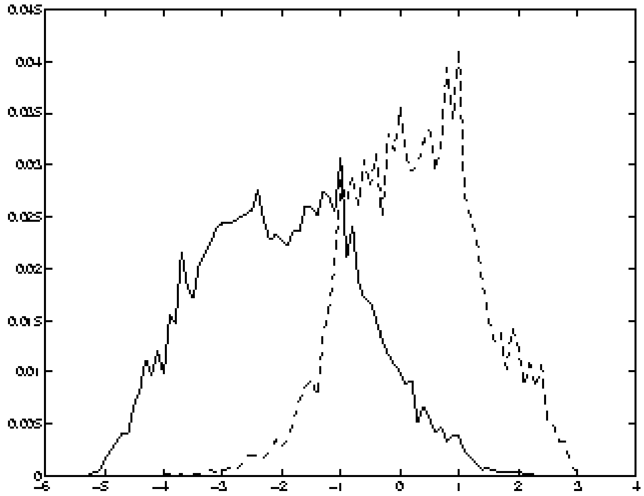

Figure 4.1: Histogram of posterior probabilities for a model trained using a linear kernel (from [91]). Note the non-Gaussian distributions.

Figure 4.1 is taken from [91], and does not prove the usability of this method on speech data. I have done this in figure 4.2, where a validation set has been used to train the sigmoid parameters of SVM model /aa/ vs. /ae/. The figure shows the actual posterior probabilities of the test set. As it can be seen, the sigmoid offers a rough, but decent estimate of the probability.

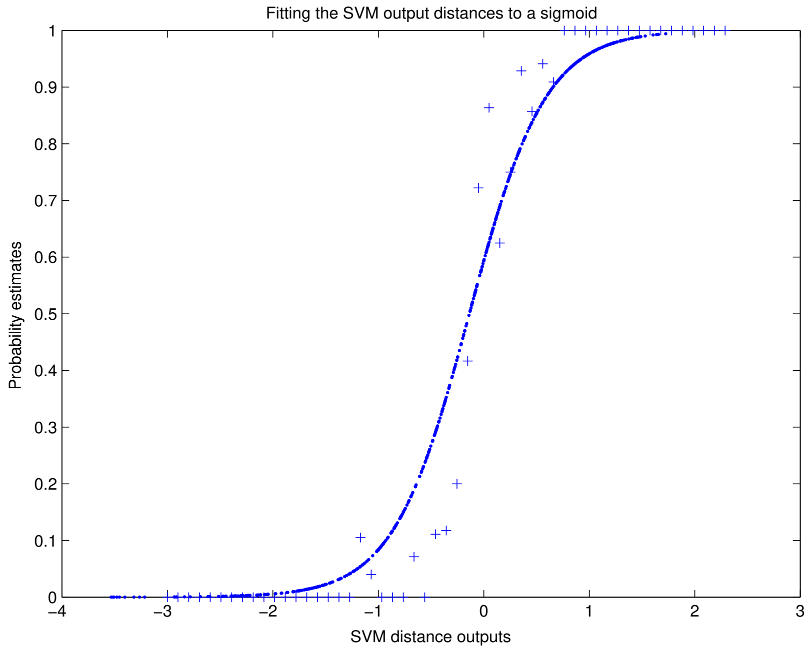

Figure 4.2: A sigmoid fit to the distance-based posterior probability estimates (/aa/ vs. /ae/).

4.2 Weighted Voting

A new multi-class classification scheme can be created to exploit the produced probability estimates. In the one vs one classifier, the voting scheme assigns binary "votes" (with value 1) to the competing classes. Instead of doing this, the 1-value voting scheme can be replaced by a probability weighted voting. Using the above method to obtain probability estimates, each class received a "vote" with a value equal to the class posterior probability $P(y = 1|x)$ or $P(y = -1|x)$.

This is formulated as:

$$f_{weighted1vs1}(x) = \arg\max_{i} {}^{K}_{i=1} {}^{K}_{j=1 \wedge i \neq j} P_{i,j}(y = i|x) \tag{4.2}$$

where K are the total number of classes, and $f_{weighted1vs1}(x)$ the predicted class.

By using this more advanced scheme, it is hoped that it will improve upon the performance of the one-vs-one scheme.

4.2.1 Experiments

The experimental setup was identical to the setup in the last set of One vs One experiments (see section 3.1.3). Experiments were carried out, and the results are found in table 4.1.

Examples per class	MBA Val	MBA Test	Accuracy	Top 3/Top 5
1000	95.5	95.1	38.6	52.1/82.7
4000	96.1	95.6	38.7	54.8/83.6
16000	96.5	**96.1**	49.9	**61.4/85.1**
16000 (full)	96.5	**96.1**	**54.3**	61.3/85.1

Table 4.1: Performance of using the weighted voting scheme.

Perhaps surprisingly, the results from using this weighted voting scheme are far poorer than the previous results. However, a reason for this can be found. One explanation is that the estimates of the posterior probabilities are poor. This may be true, since the estimates are based on distance outputs from a maximum margin classifier with no (clear) link to Bayesian theory.

However, even though the posterior probability estimates may be poor, they should not cause such a big decrease in performance. Another explanation can be found in the unusual high number of *false positives* from a few influential classes. These classes consistently "win" most classifications with probability $P(y = 1|x)$ close to 1. In the One vs one scheme above, the irrellevant votes[1] was distributed fairly uniformly. This is not the case here. Even if the correct classes win all/most of their votes, they do have some "hard" decisions where the test samples lies close to the decision boundary. With the weighted voting scheme, a class that correctly wins *two* "hard" classifications will only get a "reward" just above 0.5. This results in almost the same "reward" as another class winning *one* "easy" SVM with a large margin (i.e. "reward" close to 1). This scheme favors very different phonemes such as nasals, since they win most of their votes by a large margin.

4.3 Conclusion

The extension of making the SVM produce posterior probability estimates is important and necessary if the method is to be used in connection with a full speech recognition system. The method described in the chapter maps the distance-outputs produced by SVM classifications to posterior probabilities by means of a sigmoid function that has been fit to the data. This was shown to produce decent, although fairly rough, estimates.

In the experiment that attempted to advantage of the probability estimates, the confidence measures were used to weight the votes in the One vs One voting scheme. Unfortunately, this was shown to degrade performance.

[1] I.e. votes from classifications of classes that are not one of the two classes involved in the binary classifier.

Part II: The Kernel Fisher Discriminant

Chapter 5

New Kernel-based Methods

This second part of the thesis will focus on improving upon the results found by the Support Vector Machines. Firstly, in this particular chapter, several alternative kernel-based methods will be reviewed, and one chosen for closer examination. Then in chapter 6, the chosen method, the Kernel Fisher Discriminant (KFD), will be described. Additionally, a technique for speeding up the training process will be explained. Finally, chapter 7 will report results based on experiments performed using the KFD.

5.1 An Improved Classifier

What problems are still not resolved by the SVM? Improving upon the results found in chapter 3 can be done in many ways. The following will list some of the key issues that must be solved.

- *Scalability and training time*
  As mentioned earlier, the training time of the SVM increases near-quadratically with the number of training samples, so it's use is clearly going to be problematic for large data sets. In our previous experiments[1], we only managed to train the SVM using around 40% of the TIMIT training set. This took of the order of 10^4 (10 thousand) hours of CPU time on 750 MHz UltraSparc 3 processors. For the full TIMIT training set, this results in an estimated 6 years of CPU time (!). Even with ever increasing CPU speeds, training a SVM of the type in our first experiments is not going to be practical on larger data sets in the near future[2]. A solution to the scaling problem is the key to successfully using SVMs for speech

[1] See section 3.1.3.

[2] TIMIT contains only 68 minutes of speech. In contrast, the more recent SWITCHBOARD speech corpus contains about 240 hours of speech [37]. That is over *300 million* phoneme frames.

recognition.
The scalability not only involves reducing training time, but also includes memory issues that arises when dealing with large data sets. An improved classifier should have training time that scales linear or near-linear ($O(N^2)$) with amount of training data N. Moreover, it must have low memory requirements, that are kept low even for large training sets.

- *Sparsity and classification time*
  Another important factor is the sparsity of the solution created. Since classification time (i.e. number of evaluations) are linearly related to the number of support vectors in the model, the classifier should create models that contain as few support vectors as possible. Note that this should be done without compromising prediction accuracy too much, if at all.

- *Probabilistic outputs*
  As mentioned in chapter 4, if any kernel based method is to be used in connection with a decoder and a full recognition system, it will *need* to be able to produce class probability estimates. This allows the model to be a direct replacement for the neural network in systems like [6] and [93]. The SVM was able to produce probability estimates when using a fitting method (see section 4). However, as seen from the results, those estimates were only appriximate. An improved classifier should produce better estimates, and, at best, should produce output that can be *directly* interpreted as class conditional probability estimates.

- *Online training*
  It is useful that the classifier is able to be trained incrementally, so it does not require all training data to be available at once (i.e. online training as opposed to batch training). This is useful when training on the massive speech data sets that exist. Moreover, it can be useful in a practical speech recognition system. To personalise this, in most personal speech systems the speaker can retrain the model and adapt the system to his or her voice. This should also be possible for the phoneme recogniser, and naturally without completely retraining the model.

Obviously, it is unlikely that a classifier exist which fulfills all the above attributes. The search should concentrate on finding an improved method that fulfill as many of the listed attributes as possible.

5.2 How to improve

There are two approaches to the problem of improving upon the SVM:

1. Improving the SVM method itself
2. Finding a new kernel-based method

Regarding the first option, extensive work has been done to improve upon SVMs. Most of these have focused on speeding up training time or optimising the SVM hyper-parameters. The improvements in speed are focused on: a) Improvement upon existing decomposition methods [104, 68], b) New decomposition algorithms [60, 67], and c) speeding up the training process by finding approximate solutions by reducing the problem size [35, 31, 108, 66, 124][3]. The problem of producing class conditional posterior probabilities has been examined in [91], [109] and [63].

However, even though a lot of improvements have been made from the conception of the original SVMs, the results have been numerous small improvements. As a consequence, the actual *practical* improvements in accuracy, training speed, etc in the last 4 years have been limited. As probably the best proof of this, the best results on several public data sets are not found by SVMs anymore, but (often) by other kernel-based methods[4]. In addition, another indication that SVMs may not be the way forward is that many leading researchers, including the inventor Vapnik himself, have moved on to research alternative methods.

For these reasons, I have opted to search for an alternative kernel-based method.

5.3 New Kernel Machines

The number of new kernel machines that have appeared following the introduction of SVMs makes it impractical to describe them all in detail. Fortunately, a lot of methods can be ignored based on the criterias made in the above list. As mentioned earlier, for any classifier to be incorporated into a speech recogniser it *must* be able to produce class conditional posterior probabilities. In the previous chapter, Platt's method was used for moderating the distance based SVM outputs [91]. This method was not by any means optimal. What is needed is a kernel method whose outputs can be directly interpreted as posterior probabilities.

This requirement eliminates a lot of kernel methods that otherwise have shown to have excellent scaling and generalisation performance (some of the most promising

[3] A good summary of the work done to improve speed can be found in [115].
[4] For an example, see [81].

being the Lagrangian support vector machine [75], the Proximal Support Vector Machines [30], and a fixed point algorithm [62]). However, there are still multiple kernel machines left that does fulfill the requirement.

5.4 Bayesian-based Kernel Machines

Bayesian-based kernel machines have the needed ability to produce class conditional probability estimates from classification outputs. In this section, the ones deemed most applicable will be reviewed. The selected methods will be briefly explained and their advantages and disadvantages evaluated with reference to the list of attributes in section 5.1.

Practical issues should also be considered. Some of the methods may look promising or have good results on smaller data sets, but to apply them to a large scale multi-class problem, they must be able to process millions of samples. This requires a solid and stable software implementation. Some promising Bayesian methods [52, 2] had to be dismissed because their implementations were unatainable or unstable. In these cases, I could have proceeded to implement them myself, but due to time limitations this was not an option. Therefore, I only examined methods where the authors implementation were available or that were easily implementable in Matlab[5].

The following will describe the four kernel methods I chose to examine in details.

5.4.1 Least Squares Support Vector Machines

The first method to be considered is the Least Squares Support Vector Machine (LS-SVM). It is a mathematical reformulation of the SVM method [111]. Due to it's new formulation it was possible to add a Bayesian framework to the formulation [24].

The difference between the basic LS-SVM and the SVM is not that large. Recall the SVM formulation use inequality constraints in equation (2.21) in section 2.3.2:

$$\begin{aligned} \mathbf{w} \cdot \mathbf{x}_i - b \geq +1 - \xi_i \quad for \quad y_i = +1 \\ \mathbf{w} \cdot \mathbf{x}_i - b \leq -1 + \xi_i \quad for \quad y_i = -1 \end{aligned} \tag{5.1}$$

where $\mathbf{w}$ is the weight vector, $\mathbf{x}_i$ the training samples, y_i the corresponding targets, b the threshold and $\xi_i theslackvariables.TheLS-SVMchangesthistooneequalityconstraint:$

In addition, a quadratic penalty term ξ_i^2 is put on misclassifications in equation (2.22) instead of the SVM's ξ_i. This allows a simple formulation of the problem as a set of linear equations.

[5] However, I have dedicated a small section below for describing some of the methods that I felt should have been examined too (see section 5.4.5)

The equality term closely relates the LSSVM formulation to the Kernel Fisher Discriminant, another kernel machine which will be described in section 5.4.4. Both methods attempt to fit the data from each class to the class labels +1 and −1, and can be seen as two seperate regressions performed on the class labels. However, in contrast to the KFD formulation, here the penalty term is ξ^2 and not ξ, giving rise to the name "Least Squares SVM".

Bayesian interpretation of the LS-SVM formulation The LS-SVM can be set in a probabilistic framework, by adding an "extra layer" on top of the current formulation [24]. Applying Bayes' rule at 1st level of inference, that is, maximising the likelihood $p(y|\boldsymbol{\alpha}, \sigma^2) * p(\boldsymbol{\alpha}|C)$, the class conditional posterior probabilities can be found[6]. In addition to this first layer of inference, a second and third layer of inference can be used to find, respectively, optimal penalty term and kernel-parameters.

The Algorithm Training the LS-SVM system involves solving a linear programming problem. This can be done efficiently using existing linear programming packages. The resulting training time scales approximately $O(N^2)$ and memory requirements $O(N^2)$ with the number of training samples N. However, memory requirements can be reduced by using an iterative on-line technique from [3] and computational requirements for calculating the layers of inference can be reduced [1].

Advantages and disadvantages The main advantage of the LS-SVM is the added ability to interpret the predicted outputs as class conditional posterior probability estimates. In addition, the performance of the LS-SVM has been shown to outperform SVMs on a large set of problems [111]. Finally, training time is claimed to be slightly faster than SVMs[7].

However, the LS-SVM also has it's problems. Mainly, the current formulation of the classifier creates *dense* solutions, i.e. all $\alpha_i \neq 0$ and therefore all training samples are used as support vectors. This makes classification very slow. To fix this problem, a post-processing pruning was proposed in [70] where α_is closest to zero are removed from the solution. Although this reduces model performance and requires more processing time, it was shown to still outperform SVMs using only 15% of all training samples as support vectors.

[6] See [24] for a detailed deriviation.

[7] This claim was not verified in experiments: the training speed was the same as the SVM. This could perhaps be due to the fact that the LS-SVM was implemented in C++ but used Matlab as a front-end (the two processes share resources which makes training slower).

5.4.2 Bayes Point Machines

The Bayes Point Machine [49] is a Bayesian-based kernel method, which uses sampling to obtain class posterior estimates in classification. It is quite different from both SVMs and LS-SVMs. The idea behind the method lies in the calculation of the *Bayes Point:* the only solution, in terms of weight vector $\mathbf{w}$, that yields a Bayes-optimal decision boundary. [47].

To explain more precisely what the Bayes Point is, we define V as the set of weight vectors $\mathbf{w}$ that all separate the training samples perfectly, i.e. region in hypothesis space with zero training error. This can be written as:

$$V = \{\mathbf{w} \mid y_i \cdot f(\mathbf{x}_i) > 0; i = 1, \dots, n, \|\mathbf{w}\|_2\} \tag{5.2}$$

where $f(\mathbf{x}_i)$ is the prediction of sample i and y_i is it's corresponding target[8]. This set V is also called the *version space*, and can be visualised as seen on figure 5.1.

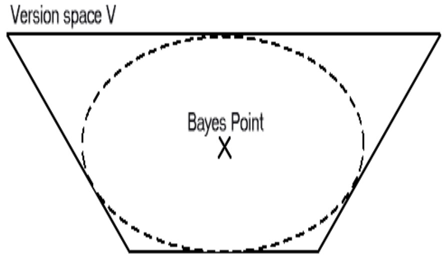

Figure 5.1: Visualisation of the version space and Bayes Point. The region outlined is the version space. The "X" inside it is the Bayes Point, which is the symmetry centre of the largest inscribed ellipse inside the version space.

However, the question is which weight vector $\mathbf{w}$ is the optimal one? This is where the Bayes Point is introduced: To be exact, it is the *centre of mass* of the version space (pictured as the cross on figure 5.1).

However, it is generally very hard to compute the Bayes Point exactly as we need to know the shape of the version space to do so (and this shape is hard to find as we need to know all possible $\mathbf{w}$s). Instead, we use an approximation and define the Bayes Point as *the symmetry center of the largest inscribed ellipse* in version space.

There are several ways to find this symmetry center. Again, since it is also computationally hard to find the exact largest ellipse in version space, sampling is used to estimate the ellipse. The Bayes Point will therefore simply be the centre of all samples. Methods include Gibbs sampling [38], a Monte-Carlo style billiard algorithm

[8]Note that in this equation we have set the bias $b = 0$.

[49] and the well-known perceptron algorithm [46]. Although the two first ones have been shown to produce excellent results, in most cases even outperforming SVMs, their training time scale $O(L \cdot N^2)$ and memory requirements $O(N^2)$, where N is the number of training samples and L the number of estimated classifiers used for sampling[9]. Fortunately, the very simple perceptron algorithm is much cheaper, scaling only $O(L \cdot N)$ in training time and $O(N)$ in memory requirements. In addition, large scale results from [46] show that it's performance is similar to SVMs.

The Algorithm Consequently, the variation of the Bayes Point Machine used in the examinations employed the perceptron algorithm to find different classifiers in version space (i.e. different weight vectors $\mathbf{w}$). The algorithm is run on small subsets of the training data, where each finds a different $\mathbf{w}$ by repeatedly changing the weight $\mathbf{w}$ until *all* training samples are correctly classified by $\mathbf{w}$. In other words: when the vector $\mathbf{w}$ corresponds to a point in version space.

Classification As mentioned above, classification is done by sampling over the L classifiers. Each classification is done as in the SVM section (see 2.4.3). However, the real-valued outputs $f(\mathbf{x})$ are used, and the final classification is the average of all samples. This gives an estimate of the class-conditional posterior probability as seen in [49].

Advantages and disadvantages The main advantage of the Bayes Point Machine is that it is has both a computationally cheap training cycle, and can create acceptable posterior probability estimates. In addition, experiments have shown that it has comparable performance with the SVM.

However, it is not without it's problems. First of all, the classification process is slow. As L classifiers are needed to perform sampling, each unknown test sample must be evaluated on all classifiers. If L is high, classification time will be high. Secondly, there is a potential problem regarding the version space. If the version space is irregularly shaped and cannot be properly estimated by an ellipse, the estimated Bayes Point *will not* be the centre of mass in version space, and therefore a bad estimate of the actual Bayes Point. Finally, the current formulation does not take overlapping class data into consideration. Although this can be done by introducing

[9] To obtain proper estimates, experiments from herbrich00large have shown that N should be at least 10.

slack variables as in the SVM formulation (see [48]), but by doing so it changes the basic idea of the version space[10].

5.4.3 Gaussian Process Prediction

The third approach to be considered are Gaussian Process Prediction. Initially, Gaussian Processes were used for regression [73]. However, recently they has been extended to handle classification problems [122].

The principle behind the method is to make assumptions about the data that are so specific that they can used to discriminate between classes. This is done by placing a prior probability distribution over the possible functions $f(\mathbf{x})$ that fits the data $\mathbf{x}$ to their targets y. The training process involves "fitting" this prior into a posterior using the training data. In other words, the training data is used to "learn" assumptions on the data, and will be used to find a *prior*, which in turn will change the resulting *posterior*.

The Algorithm From the pool of possible assumptions that can be put on the data, the discriminate prior is found by removing the priors that are inconsistent with the training data. This problem can be formulated as a linear program that can be solved. However, in this process a large $N \times N$ eigendecomposition must be performed, where N is the training set size. This results in a training time of $O(N^3)$, which is infeasible for larger data sets.

Fortunately, this expensive step can be avoided by a numerical approximation technique called the *Nyström method* [124]. This technique solves the eigendecomposition on a much smaller system of size M, and then expands the results back up to N dimensions. This reduces training time to $O(N \cdot M^2)$ and memory requirements to $O(N \cdot M)$. Although the approach only finds an approximate solution, even low values of M have been shown to produce solutions with performance close to that of the full solution[11] [124].

Advantages and disadvantages Gaussian Processes prediction could potentially be very effective on the task in focus. They can produce class conditional posterior es-

[10] I.e. that the version space is the set of $\mathbf{w}$s with zero training error. If the classes are overlapping this set will be empty. Instead, the Bayes Point is estimated as the point in version space, which minimises the training error the most.

[11] Values as low as $M = 256$ was found to produce models with acceptable performance for an example with N=2000. See [124] for details.

timates, and easily be extended to the multi-class case[12]. In addition, the performance of the method compares favourably with the SVM results.

However, the method has some problems. The prior is estimated using combinations of Gaussians that assumes the training data and noise can be estimated by Gaussians. It assumes that the data can be fitted by a smooth and continuous function, which is not always the case[13].

5.4.4 Kernel Fisher Discriminant

The Kernel Fisher Discriminant (KFD) is a version of Fisher's classical linear discriminant extended to incorporate kernels [80]. The basic principle behind the method is an attempt to find a decision boundary (in terms of support vectors) that both *maximises the distance between the class means* and *minimises the within-class variance.*

The Algorithm The method can be written as a quadratic program very similar to the one found in SVMs, and can be solved efficiently by a Sparse Greedy Approximation technique [108, 82]. This is an iterative technique that starts with an empty solution and adds one support vector per iteration until the value of the objective function in the quadratic program reaches a predefined threshold. At each iteration, a predefined number of training patterns are examined, and the one that best minimises the expression is added to the set of support vectors.

The resulting *Sparse Greedy KFD* reduces scaling to $O(N\ M^2\ Q)$ and memory requirements to $O(N\ M^2)$, where N is the training set size, M the number of support vectors in the final solution, and Q the number of training patterns examined in each iteration. If the required number of support vectors is low, this method completes training much faster than the SVM and with lower memory requirements, while still producing comparable performance (see [82]).

Classification As with all kernel machines considered in this section, classification is done in the same manner as SVMs, i.e. the classifier can be written as a linear combination of support vectors.

Advantages and disadvantages In addition to a fast training cycle, a significant advantage over SVMs is inherited from Fisher's discriminant: assuming Gaussian class distributions, the outputs of a KFD can be interpreted as class-posterior probabilities. Moreover, the KFD has shown to produce extremely sparse solutions, using only a

[12] See [122].

[13] Fortunately, preliminary results on the TIMIT data set seemed to work well and not have this problem.

fraction of the support vectors of SVMs. This results in a similar reduction in classification time.

However, a main problem with the method is the severe assumptions it puts on the data. The data of the classes are assumed to be distributed as two Gaussians with equal covariance matrices. This assumption does not always hold, and in these cases the KFD performs poorly.

5.4.5 Other Alternatives

Plenty of other promising kernel machines are possible. Among these are *Kernel Principal Component Analysis* [97], and *Bayesian Committee Machines* [103]. Although these methods also promise various improvements upon SVMs, they were not examined in detail because of either lack of a proper implementation and/or inability to handle large data sets.

However, one very promising method was perhaps dismissed too quickly: The Relevance Vector Machine (RVM) [114]. This method was not examined in detail, but very late in the project period, I came up with a few ideas of how to apply it to the problem.

The Relevance Vector Machine This is a probabilistic approach very similar to the Kernel Fisher Discriminant and the Least Squares SVM. It can be viewed as a normal SVM with an added Bayesian layer. The extra layer attempts to find the values of hyperparameters that maximises the likelihood of the training data. A detailed description of the RVM can be found in [114]. Several independent experiments in [14], [12] and [44] have showed the method to produce extremely sparse solutions with excellent generalisation performance[14]. Together with the automatic optimisation of hyperparameters, this creates a very powerful classifier.

However, as expected the classifier has it's drawbacks. During the training process, a matrix inversion is required, resulting in a training time which scales $O(N^3)$ with the number of training samples NTherefore, the RVM is far too slow to be used for the problem in focus.

[14]The last reference is interesting, and was what lead me to consider RVM again. It is from a PhD Thesis proposal, where RVM are applied to Speech Recognition problems. Preliminary experiments on a small speech data set were very promising and suggests that RVM have a great potential on speech related tasks. However, as the PhD only started few months ago, no publications have been made yet.

Thoughts on the RVM The Nyström method [124] can be used to speed up the training process 5.4.3. This would result in a training time that scales of $O(L \cdot N^2)$ with the training set size N, where L is a fixed number[15]. Even though this would still not enable the RVM to use all training data, it would make the method feasible given it's superior performance compared to other methods.

Another possibility was to use ensembles of RVMs trained on smaller, and manageable, subsets of the training data [26]. Classification would then sample the scores found by each classifier to produce a final prediction. This technique has shown to produce comparable results to creating a single model from all the training data [26]. The drawback of this approach is a slower classification process due to the fact that each unknown test sample must be evaluated on all machines in the ensemble. However, the extreme spasity of the RVM models would make up for this.

5.5 Selecting a method

As time restrictions did not allow me to perform extensive experimentation on all four methods, one classifier from the list above had to be chosen. However, I did perform preliminary experiments with each method to verify claims of scaling factors, probabilistic outputs etc. Comparing with the list in section 5.1, the following theoretical and practical issues were found:

- *Scalability and training time*
  All methods claim to improve upon the training speed of the SVM. However, one must look at *how* much each method reduces the scaling factor. Although TIMIT contains approximately 1.5 million samples it is a small data set compared to more recent speech recognition corpora such as Switchboard. For a method to be useful on proper speech problems it must be able to handle significantly more training data than SVMs. From the above methods, the Bayes Point Machine and the Kernel Fisher Discriminant have the best scaling attributes.

- *Sparsity and classification time*
  As mentioned before, although fast classification is not the specific focus of this thesis, it cannot be ignored. Creating sparse solutions is important making kernel methods practical. Except for the Least Squares Support Vector Machine, all the above methods are able to create sparse solutions. However, where the Bayes Point Machine generates multiple classifiers and uses sampling to obtain a prediction, the two "fastest" methods are Gaussian Processes and Kernel Fisher

[15] $l = 256$ has shown to produce good results in [124].

Discriminant. Gaussian Processes creates mixtures of Gaussian functions for use in prediction, and preliminary experiments showed that the Kernel Fisher Dicriminant creates extremely sparse solutions, needing only a fraction of the support vectors of SVM models.

- *Probabilitic outputs*
  The quality of the posterior probability estimates are also important. As mentioned earler, the ability to produce proper estimates are a requirement for using the method in a full speech recognition system. One could be worried about the approximations made by the Kernel Fisher Discrimiant (assumption on the Gaussian distribution of the data), and the Bayes Point Machine (since rough sampling is used to estimate the Bayes Point). However, practical results showed that all methods seemed to produce convincing and produced almost identical results.

- *Online training*
  A flexible, incremental, training process is desired to enable future updates to a model without retraining it completely. The only method to have this ability was the Kernel Fisher Discriminant. It added a training sample to the solution per iteration until an optimal solution was reached. However, the remaining

- *Implementation*
  A factor to consider, which was *not* included on the list in section 5.1 is the quality of the implementation. Although this is not a theoretical issuea, it is at least as important. Here, the KFD and GP had very stable C/C++ implementations. Moreover, the KFD used matlab format to store data and save results, making it easy to analyse results. The Least Squares Machine was written as a C++-implementation with Matlab as front-end, making it slightly slower than the two methods above as resources had to be shared with Matlab. The Bayes Point machine implementation used was written in R, which made it significantly slower.

Based on the above observations, either the Bayes Point Machine or the Kernel Fisher Discriminant seemed to be best suited for the problem. This conclusion was reached when weighting the importance of being able to handle large data sets. Of the two methods, two concerns of the BPM made me select the KFD method. Firstly, the implementation in R was not very fast, leading to a higher than necessary training and classification times. I could have opted to implement the method in C++, but this would have required a substantial amount of time. In contrast, the KFD implementation was already a fast C++ implementation. Secondly, and more importantly, in [48] the BPM was shown to have problems with classes with high overlap. As mentioned above, if classes have high overlap, there are no weight vectors to create the version space (the "zero training error region"). Perhaps the method for solving this problem in [48] is not optimal.

Based on these conclusions, the Kernel Fisher Discrimiant was chosen for the final experiments. The usefulness of this choice was confirmed in preliminary results where it showed to perform very well (explained later in experiments chapter).

Chapter 6

The Kernel Fisher Discriminant

In this section, the Kernel Fisher Discriminant (KFD) will be explained. This kernel machine, introduced by Sebastian Mika[1], is a version of the classical Fisher linear discriminant extended to utilise kernels. Combined with a technique to speed up training termed the Sparse Greedy Approximation technique, the claimed advantages are many [82]: It should have a training time which scales linearly with the number of training samples, potentially completing training much faster than SVMs and with lower memory requirements. Moreover, it should create extremely sparse solutions with performance comparable to SVMs. Finally, because of the iterative approach, training can be performed "on-line" to enable easy updating of the model if new training data becomes available.

The section will begin with a review of the most recent KFD literature. Subsequently, the method itself will be carefully described, and finally, the Sparse Greedy Approximation technique will be explained and applied.

6.1 Review of recent literature

The KFD method has previously been applied to a few problems, where it have shown promising results. Sebastian Mika introduced the method in [80]: in this article he described the basics of the method, and how the basic linear discriminant is extended to make use of kernels. Results reported from experiments on several public data sets showed the feasibility of the method: the results were equal or in most cases *better* than Support Vector Machines.

In [79], further investigations were made. Work was done to create sparse KFD solutions, and links to a Bayesian framework and other popular kernel machines were

[1]Fraunhofer Institute FIRST in Berlin, Germany, mika@first.gmd.de

explained and examined. Results showed the method to produce excellent class posterior probabilities. Finally, a kernel-regression technique based on the KFD was outlined.

[125] introduced a Minimum Squared Error kernel machine. It was proven that the solution created by this method was identical to the solution of the KFD in terms of decision boundary except for a scaling factor. Comparisons in experimental testing showed that both method performed well on difficult problems.

In [81], an attempt was made to incorporate prior knowledge into kernel methods. One of the examined classifiers was the KFD. Experiments performed on a handwritten digits data set showed the KFD to have one of the best ever results reported on the data set.

Another practical application of the KFD was reported in [8]. Here, both the KFD and SVMs were applied to the problem of extracting semantic information from images. Results showed that both kernel methods performed similarly, and significantly better than linear methods. It was noted that KFD had a significant advantage over SVMs since it could produce accurate class-conditional density estimates.

A third real-world application was made in [23]. Once more, the chosen methods were KFD and SVMs. The task was to identify a chemical substance's type based on it's "electronic nose signature". Comparisons with more traditional methods were done, and it was found that the kernel methods offered a lot of promise. However, a less optimal multi-class scheme was used, and thus resulting in slightly worse results than the traditional methods.

Finally, in [64], the potential of the KFD's class-conditional posterior estimates was exploited to handle data with noisy labels, i.e. training data where the targets y were not always correct. An algorithm was proposed that was shown to improve the robustness slightly.

The above examples all show the potential of the KFD as a non-linear discriminant classifier. Except for a single case ([23]) the KFD proved to have excellent generalisation abilities, while being able to produce high quality class posterior estimates from classification outputs.

6.2 The Kernel Fisher Discriminant

In this section, the concepts behind the KFD will be explained. KFD is based on Fisher's linear discriminant, and extended to a non-linear discriminant by utilising a kernel. The idea of "kernelising" an existing method was explained in section 2.4. To summarise, instead of finding a linear solution in *input space*, the "kernel trick" is performed (see section 2.4.1). Using a kernel, the linear classifier is applied to the data in *feature space* instead of the data in input space. If the kernel's projection from input to feature space is non-linear, this will result in a non-linear decision boundary in input space.

The requirements for applying the "kernel trick" to a linear classifier are fairly simple: All that is needed is a rearrangement of the mathematical formulation such that input data is introduced only as it's inner product $\mathbf{x} \cdot \mathbf{x}$. In some cases this may not be possible, but to date the idea has been applied to several existing methods, such as Principal Component Analysis [97], Logistic Regression [116] and Generalised Linear Discriminant [95, 5]. The following section will describe how Fisher's linear discriminant is defined, and how it is extended to utilise kernels[2].

6.2.1 Fisher's Linear Discriminant

Fisher's discriminant aims to find a linear decision boundary such that the classes are well separated. The measure of separability is defined by two quantities:

1. *Distance between the class means* Σ_B. This should be as *large* as possible.
2. *Within-class variance* Σ_W. This should be as *small* as possible.

How the discrimiant models the class data and how this effects the margin is shown on figure 6.1.

In the following, S_+ and S_- will denote the two sets of samples from each class with set sizes $|S_+| = N_+$ and $|S_-| = N_-$. Moreover, $\mathbf{m}_+$ and $\mathbf{m}_-$ will defined the means of the two classes such that:

$$\mathbf{m}_+ = \frac{1}{N_+} \sum_{x \in S_+} x \quad \text{and} \quad \mathbf{m}_- = \frac{1}{N_-} \sum_{x \in S_-} x \tag{6.1}$$

Separating the classes is achieved by maximising the so-called *Rayleigh coefficient*[80]:

$$J(\mathbf{w}) = \frac{\mathbf{w}^\top \Sigma_B \mathbf{w}}{\mathbf{w}^\top \Sigma_W \mathbf{w}} \tag{6.2}$$

where $\mathbf{w}$ is the weight vector to be found, and:

$$\Sigma_B = (\mathbf{m}_+ - \mathbf{m}_-)(\mathbf{m}_+ - \mathbf{m}_-)^T \tag{6.3}$$

[2] The following description is based on [82, 79, 80, 24].

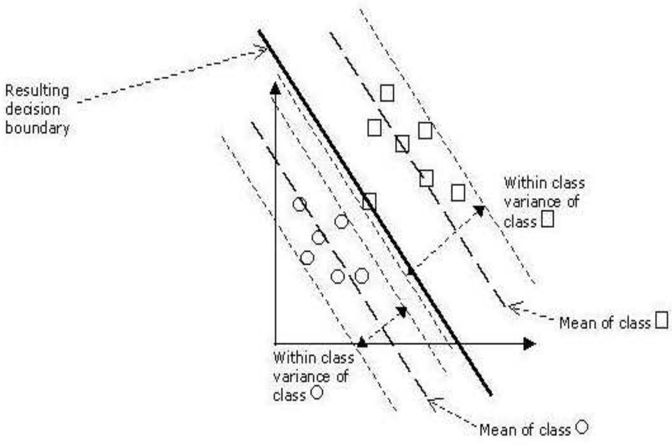

Figure 6.1: Classification using Fisher's linear discriminant. The decision boundary is defined in terms of the class means and within class varances. The long stipled lines shows the two class means, and the short stipled lines attempts to visualise the within-class varience. Where the resulting decision boundary is put between the classes are affected by both factors.

is the (squared) distance between the class means. Σ_W is the sum of the within-class variances:

$$\Sigma_W = \sum_{i \in S_+} (\mathbf{x}_i - \mathbf{m}_+)(\mathbf{x}_i - \mathbf{m}_+)^T + \sum_{i \in S_-} (\mathbf{x}_i - \mathbf{m}_-)(\mathbf{x}_i - \mathbf{m}_-)^T \tag{6.4}$$

In other words, maximising the Rayleigh coefficient $J(\mathbf{w})$ is an attempt to find a weight vector $\mathbf{w}$, which maximises the distance between class means while minimising the within class variance.

6.2.2 Kernel Fisher Discrimiant

The extension to the non-linear case is done by applying the kernel trick (see section 2.4.1). This needs reformulation of equation 6.2 such that input patterns $\mathbf{x}_i$ and $\mathbf{x}_j$ are solely represented in terms of a functional $K(\mathbf{x}_j, \mathbf{x}_i) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$.

To do this, we can formulate the weight vector $\mathbf{w}$ in terms of the mapped training patterns:

$$\mathbf{w} = \sum_{i=1}^{N} a_i \Phi(\mathbf{x}_i) \tag{6.5}$$

where $\Phi(\cdot)$ is the mapping from input space to feature space.

By defining the *projected class means*:

$$\mathbf{m}_{+}^{\Phi} = \frac{1}{N_{+}} \sum_{x_i \in S_{+}} \Phi(\mathbf{x}_i) \tag{6.6}$$

and

$$\mathbf{m}_{-}^{\Phi} = \frac{1}{N_{-}} \sum_{x_i \in S_{+}} \Phi(\mathbf{x}_i) \tag{6.7}$$

We can combine 6.5 with 6.6 and 6.7 to isolate the mappings $\Phi(\cdot)$ and utilise the kernel function:

$$\mathbf{w}^{\top}\mathbf{m}_{+}^{\Phi} = \frac{1}{N_{+}}\sum_{j=1}^{N}\sum_{k=1}^{N_{+}}\alpha_j \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j) = \frac{1}{N_{+}}\sum_{j=1}^{N}\sum_{k=1}^{N_{+}}\alpha_j K(\mathbf{x}_j,\mathbf{x}_k) \tag{6.8}$$

$$\mathbf{w}^{\top}\mathbf{m}_{-}^{\Phi} = \frac{1}{N_{-}}\sum_{j=1}^{N}\sum_{k=1}^{N_{-}}\alpha_j \Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j) = \frac{1}{N_{-}}\sum_{j=1}^{N}\sum_{k=1}^{N_{-}}\alpha_j K(\mathbf{x}_j,\mathbf{x}_k) \tag{6.9}$$

Now, if we define two new terms $\mathbf{M}_{+}$ and $\mathbf{M}_{-}$:

$$(\mathbf{M}_{+}) = \frac{1}{N_{+}}\sum_{j=1}^{N}\sum_{k=1}^{N_{+}} K(\mathbf{x}_j,\mathbf{x}_k^{+}) \tag{6.10}$$

$$(\mathbf{M}_{-}) = \frac{1}{N_{-}}\sum_{j=1}^{N}\sum_{k=1}^{N_{-}} K(\mathbf{x}_j,\mathbf{x}_k^{-}) \tag{6.11}$$

Then we can rewrite 6.8 and 6.9 as:

$$\mathbf{w}^{\top}\mathbf{m}_{+}^{\Phi} = \boldsymbol{\alpha}^{\top}\mathbf{M}_{+} \tag{6.12}$$

$$\mathbf{w}^{\top}\mathbf{m}_{-}^{\Phi} = \boldsymbol{\alpha}^{\top}\mathbf{M}_{-} \tag{6.13}$$

Also, recall that the goal is to rewrite the Raleigh equation (6.2) using kernel functions. The numerator in the equation is $\mathbf{w}^{\top}\Sigma_B\mathbf{w}$. If we define:

$$\mathbf{M}_B = (\mathbf{M}_{+} - \mathbf{M}_{-})(\mathbf{M}_{+} - \mathbf{M}_{-})^{\top} \tag{6.14}$$

then it can be rewritten as:

$$\mathbf{w}^{\top}\Sigma_B\mathbf{w} = \boldsymbol{\alpha}^{\top}\mathbf{M}_B\boldsymbol{\alpha} \tag{6.15}$$

Similarly, if we define $\mathbf{M}_W$ as being the sum of *projected* within-class variances in feature space:

where $\mathbf{I}$ is the identity matrix and $\mathbf{1}_+$ and $\mathbf{1}_-$ are two $1 \times N$ matrices defining the N training samples' membership of the classes. That is, $(\mathbf{1}_+)_i = 1$ if sample $\mathbf{x}_i$ belongs to class + and $(\mathbf{1}_+)_i = 0$ otherwise. Similarly with $\mathbf{1}_-$.

Then the denominator can be written as;

$$\mathbf{w}^\top \Sigma_W \mathbf{w} = \boldsymbol{\alpha}^\top \mathbf{M}_W \boldsymbol{\alpha} \tag{6.16}$$

Finally, the two equations 6.16 and 6.15 above can be combined to produce the *kernelised Raleigh coefficient*:

$$J(\boldsymbol{\alpha}) = \frac{\boldsymbol{\alpha}^\top \mathbf{M}_B \boldsymbol{\alpha}}{\boldsymbol{\alpha}^\top \mathbf{M}_W \boldsymbol{\alpha}} \tag{6.17}$$

6.2.3 Quadratic Program formulation

Whereas the goal of Fisher's linear discriminant is to find a weight vector $\mathbf{w}$ that maximises equation 6.2, the goal of the non-linear Fisher discriminant is to find an $\boldsymbol{\alpha}$ which maximises $J(\boldsymbol{\alpha})$.

This can be done in several ways. One possible way is to solve the eigenproblem $\mathbf{M}_B \boldsymbol{\alpha} = \lambda \mathbf{M}_W \boldsymbol{\alpha}$ and choose the $\boldsymbol{\alpha}$ with maximal eigenvalue λ. There exist many optimised and effective packages that can solve this problem. However, this is only feasible for small data sets, as the sizes of both $\mathbf{M}_B$ and $\mathbf{M}_W$ scale with the number of training samples. In addition, the solutions found will be dense, i.e. have all $\alpha_i \neq 0$.

Another way of finding optimal $\boldsymbol{\alpha}$ is to reformulate the problem as a convex quadratic programming problem. To do this, we use the fact that the matrix with the distance between "projected class means", $\mathbf{M}_B$, is of rank 1 (i.e. simply an $N \times 1$ matrix, where N is the number of training samples). Moreover, we use the fact that any multiple of $\boldsymbol{\alpha}$ is an optimal solution to 6.17 (see [79] for details). Hence, we can set the nominator $\boldsymbol{\alpha}^\top \mathbf{M}_B \boldsymbol{\alpha}$ to a constant, and minimise the denominator to maximise $J(\boldsymbol{\alpha})$. This results in the following quadratic program:

$$\min_{\boldsymbol{\alpha}} \; \boldsymbol{\alpha}^\top \mathbf{M}_W \boldsymbol{\alpha} + C \cdot P(\boldsymbol{\alpha}) \tag{6.18}$$

$$\text{subject to:} \qquad \boldsymbol{\alpha}^\top (\mathbf{m}_+ - \mathbf{m}_-)) = 2 \tag{6.19}$$

The constraint $\boldsymbol{\alpha}^\top (\mathbf{m}_+ - \mathbf{m}_-))$ is simply the square root of $\boldsymbol{\alpha}^\top \mathbf{M}_B \boldsymbol{\alpha}$ and are set to 2 here. The term $C \cdot P(\boldsymbol{\alpha})$ is an added regularisation term, similarly to the one found in SVMs. It is used to counteract the problem of high dimensionality. Many different options are available for this term, and will be discussed later on.

6.2.4 Rewriting the Quadratic Program

The above quadratic program still has a problem: it still relies on the matrix $\mathbf{M}_W$, which is an $N \times N$ matrix that scales with the number of training samples N. This is clearly infeasible for large problems. To avoid this, we reformulate the quadratic program one more time.

Consider what Fisher's linear discriminant is trying to do: it attempts to find a decision boundary that maximises the distance between the class means, while minimising the within-class variance. To define the *kernelised* Fisher discriminant we seek a classifier that finds a decision boundary which maximises the distance between the *projected* class means, while minimising the *projected* within-class variance. The following QP program does exactly this:

$$\min_{\boldsymbol{\alpha},b,\boldsymbol{\xi}} \|\boldsymbol{\xi}\|^2 \quad + \quad C \cdot P(\boldsymbol{\alpha}) \tag{6.20}$$

$$\text{subject to} \quad K(\mathbf{x}_i, \mathbf{x}_j) + b \quad = \quad y_i + \xi_i \quad \forall i, j \tag{6.21}$$

$$\sum_{i \in \{y_i = +1\}} \xi_i = 0 \qquad \text{and} \qquad \sum_{i \in \{y_i = -1\}} \xi_i = 0 \tag{6.22}$$

where the weights $\boldsymbol{\alpha}_i$, training patterns $\mathbf{x}_i$, targets y_i and bias b are the same as in the SVM formulation, and $P(\boldsymbol{\alpha})$ the regularising function. This QP problem attempts to fulfill the two requirements in the Fisher discriminant:

- Regarding the within-class variance, look at the constraint (6.21). Together with the minimisation of $\|boldsymbol\xi\|^2$ in (6.20), this can be seen as trying to fit the values of $K(\mathbf{x}_i, \mathbf{x}_j) + b$ to the class labels y_i of samples $\mathbf{x}_i$[3]. The variable $\boldsymbol{\xi}$ will be small when the data samples are close to their target values, and can be seen as a measure of the how well the data fit their projected class means, i.e. a measure dependent on the within-class variances.

- As for the maximisation of the distance between means, this term is not explicitly represented. However, the two constraints in (6.22) assure that the sum of ξ_i of each class is 0. Since $\boldsymbol{\xi}$ can be considered a measure of variance, forcing both sums of within-class variances to be 0, consequently forces the means of each class to be equal to the because of constraint 6.21. As the targets are $+1$ and -1, respectively, this corresponds to a fixed distance of 2 between the two class means.

[3]This can also be seen as trying to classify the training samples to their target labels. $K(\mathbf{x}_i, \mathbf{x}_j) + b$ is the "classification" and y_i the corresponding training target. $\boldsymbol{\xi}$ is a measure of how close the predicted values are from the actual $\{+1, -1\}$ targets. Since a small $\boldsymbol{\xi}$ means a "classification" close to the target, $\boldsymbol{\xi}$ can also be seen as the *variance* of the data.

On figure 6.2 an example of how the KFD attempts to find a hyperplane is shown.

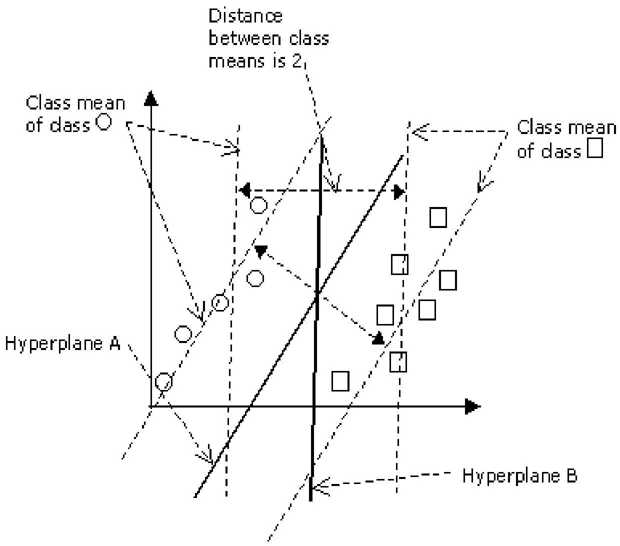

Figure 6.2: Solving the quadratic programming problem: The KFD attempts to find a hyperplane which minimise the within-class variance while keeping the distance between the margins fixed. In this case, hyperplane A will have lower within-class variance than hyperplane B.

6.2.5 Regularisation

It is very likely that the dimension of the feature space is very high - especially if the Gaussian kernel is used. This makes regularisation necessary, as the structured risk increases with dimensionality[4]. To counteract the structural risk, we have the regularisation term $C \cdot P(\boldsymbol{\alpha})$ in (6.20). The C is a penalty term similar to the one found in the SVM formulation (in equation 2.22), and $P(\boldsymbol{\alpha})$ is the regularising function.

There are several possible choices for $P(\boldsymbol{\alpha})$. The choice used in this paper is a zero-mean Gaussian, which is the same choice as used for the Relevance Vector Machine. Assuming again, that it's variance is known and a multiple of the identity, this results in a regularisation function of $P(\boldsymbol{\alpha}) = \|\boldsymbol{\alpha}\|^2$ (see [79] for proof).

6.2.6 Bayes Optimal Classifier

A major advantage of the Fisher discriminant is that the resulting weight vector $\mathbf{w}$ is an estimate of the Bayes optimal classifier for two gaussian distributions [80]. To explain the reason for this, consider a regression fit onto each of the two classes S_+

[4] As described in section 2.2. In addition, this is discussed in [102].

and S_+. These two fits will have form $(\mathbf{w} * x_i) + b$, where x_i is a pattern belonging to either S_+ or S_+, $\mathbf{w}$ is the weight vector, and b the threshold.

The posterior likelihood of the regression can be written as:

$$p(\mathbf{y}|\boldsymbol{\alpha}, \sigma^2) \cdot p(\boldsymbol{\alpha}|C) \tag{6.23}$$

where

$$p(y|\boldsymbol{\alpha}, \sigma^2) \equiv \exp(-\frac{1}{2\sigma^2}((\mathbf{w} * \mathbf{x}_i) + b - y_i)^2) = \exp(-\frac{1}{2\sigma^2}\, \|\boldsymbol{\xi}\|^2) \tag{6.24}$$

and $P(\boldsymbol{\alpha}|C)$ is some prior over the weights with parameter C

If we had $P(\boldsymbol{\alpha}|C)$, we would end up with a resulting posterior as in the Relevance Vector Machine (RVM) [113]. As mentioned in section 5.4.5, the RVM estimates the parameters C and σ^2 automatically during training, but needs to solve a very computationally expensive optimisation problem.

To avoid the expensive calculations, an approximation can be made. If we assume the gaussian noise variance σ to be known, we can remove all terms depending only on σ. By taking the log of the posterior $P(\mathbf{y}|\boldsymbol{\alpha}, \sigma^2) \cdot P(\boldsymbol{\alpha}|C)$, we get an easier optimisation problem:

$$\min_{\boldsymbol{\alpha}, b} \|\boldsymbol{\xi}\|^2 - \log(P(\boldsymbol{\alpha}|C)) \tag{6.25}$$

which is almost equal to the KFD formulation in equation (6.20). By setting $P(\boldsymbol{\alpha}|C) = \exp(\|\alpha\|^2)$, adding a penalty C to the second term, and adding the equality constraints (6.22), we will end up with equation (6.20).

This probabilistic interpretation allows the classification outputs of the KFD (in equation 6.34) as estimates of their class conditional posterior probabilities. As described in section 5.4.5, this is also the case with the RVM. Although the KFD method has links to bayesian classifiers, the advantages over the expensive RVM method is not done without costs. The KFD puts some severe assumptions on the data which are not always met. First of all, as stated above, the KFD assumes the variance of the noise to be known and the noise model to be Gaussian. Although this model can be replaced by other noise models (see [79]), this can cause problems when assumptions are not met.

6.2.7 Comparing KFD with the SVM

There are a lot of similarities between the two kernel methods. They are both inherently linear binary discriminants, where a kernel have been used to enable modelling of non-linear boundaries. However, there is one very significant difference between them. The reason for the good performance of the SVM is explained as the maximisation of the margin as being an implementation of the structural risk minimisation. Surprisingly, the KFD does *not* have an explicit concept of margin. Instead, it looks at the *distribution* of the data. While the SVM focuses on the specific training samples close to the decision booundary, the KFD focuses on class means and within-class variances. Whereas the SVM attempts to maximise the smallest margin, one could say the KFD attempts to maximise *the average margin* between the classes. This makes the KFD more robust towards outliers and noise. An example of how this difference is found in the resulting models, look at figure 6.3.

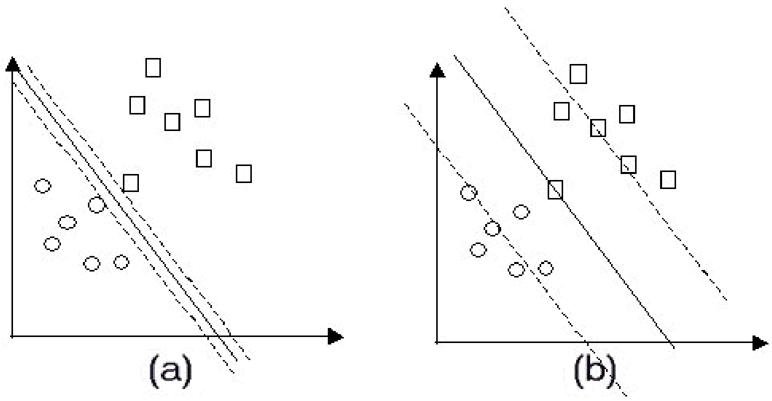

Figure 6.3: Difference in decision boundaries between the KFD and SVM. (a) shows the boundary created by the SVM. An outlier from one class results in a very narrow (minimum) margin. (b) shows the decision boundary created by the KFD. Here, the class averages are used as indication of where to put the margin, and the outlier does not have such a big effect on the final result.

6.3 Sparse Greedy Approximation

Currently, the above KFD can be solved by a quadratic programming package, scaling $O(N^3)$, where N is the number of training samples. In addition, QP packages require memory storage of $N \times N$ matrices, and this is clearly not feasible for larger data sets. Although the KFD adds the possibility of interpreting outputs as probabilities and has shown to perform competitively with SVMs [79], one of the key priorities described in section 5.1 was to improve upon the training time of the SVM. It is obvious that the current KFD formulation does not fulfill this requirement.

Fortunately, the quadratic program in expression (6.20) can be solved efficiently by a *sparse greedy approximation technique* [82]. This approach has been used to speed

Algorithm
1. Start with an empty solution with no support vectors ($M = 0$).
2. At each iteration until termination do: a. Select a number of training samples Q from the training set. b. Evaluate the selected samples to find the one that reduces the objective function in (6.31) the most. c. Add the selected sample to the solution, such that $M = M + 1$.
3. Repeat 2a-2c until a certain termination criteria has been reached.

Table 6.1: Overview of the Sparse Greedy Approximation technique

up other methods in [108, 86, 87, 107], and it is an approach along the same lines as the decomposition and chunking techniques used by the SVM software as described in section 3.0.1.

The sparse greedy approximation technique can be summarised as follows: An iterative technique adds training samples to the solution[5] until the value of expression 6.20 reaches a predefined threshold. At each iteration, a number of training patterns Q (termed the *quantile*) are examined, and the one that best minimises the value in equation 6.20 is added to the set of support vectors. In table 6.1, an overview of the algorithm is shown.

6.3.1 Applying the technique

For the sparse greedy approximation technique to work, there are four different problems that must be solved.

1. Rewriting the quadratic programming problem so a certain speed-up trick can be used (explained later)

2. Applying the trick.

3. Determining selection rules of how to select "good" support vectors, such that fast conversion is ensured.

4. Selecting a termination criterium for the algorithm

How these four problems are solved will be described now.

[5] As with SVMs, the samples in the solution are termed *support vectors*.

6.3.2 Step 1: Redefining the QP problem

The trick can be applied, the quadratic program will be reformulated. As done in [82], we define:

$$\mathbf{a} = \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix}$$

as the matrix the variables we want to estimate.

$$\mathbf{c} = \begin{bmatrix} N_+ - N_- \\ \mathbf{K}^\top \mathbf{y} \end{bmatrix}$$

as the matrix with classifications of training samples (since $\mathbf{K}^\top \mathbf{y} + \mathbf{b}$ is classification). Here, $\mathbf{K}$ is the matrix with elements $K_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$[6] and N_+ and N_- are the number of samples in S_+ and S_-.

Moreover, we denote:

$$\mathbf{A}_+ = \begin{bmatrix} N_+ \\ \mathbf{K}^\top \mathbf{1}_+ \end{bmatrix} \quad \text{and} \quad \mathbf{A}_- = \begin{bmatrix} N_- \\ \mathbf{K}^\top \mathbf{1}_- \end{bmatrix}$$

as the matrices each containing elements from $\mathbf{K}$ of each class.

Finally, we define $\mathbf{H}$ as the matrix to store the solution:

$$\mathbf{H} = \begin{bmatrix} N & \mathbf{1}_\pm^\top \mathbf{K} \\ \mathbf{K}^\top \mathbf{1}_\pm & \mathbf{K}^\top \mathbf{K} + C\mathbf{P} \end{bmatrix}$$

For the trick to be applied, the quadratic program must be reformulated such that the matrix $\mathbf{H}$ will only represented by means of it's inverse. This way, we can use a trick in the next section to compute the inverse *without ever computing* $\mathbf{H}$.

Using this terminology, the problem in equation 6.25 can be written as:

$$\min_{\mathbf{a}} \frac{1}{2}\mathbf{a}^\top \mathbf{H}\mathbf{a} - \mathbf{c}^\top \mathbf{a} + \frac{N}{2} \tag{6.26}$$

subject to:

$$\mathbf{A}_+^\top \mathbf{a} - N_+ = 0$$

and

$$\mathbf{A}_-^\top \mathbf{a} + N_- = 0$$

[6] Also termed the *Gram matrix*.

The reason for this reformulation is to enable the primal Langrangian formulation of (6.26) to be written:

$$L_P = \frac{1}{2}\mathbf{a}^\top\mathbf{H}\mathbf{a} - \mathbf{c}^\top\mathbf{a} + \lambda_+(\mathbf{A}_+^\top\mathbf{a} - N_+) + \lambda_-(\mathbf{A}_-^\top\mathbf{a} + N_-) + \frac{N}{2} \tag{6.27}$$

where λ_+ and λ_- are the Langrangian multipliers.

The resulting dual formulation is:

$$\max_{\mathbf{a},\lambda_+,\lambda_-} -\frac{1}{2}\mathbf{a}^\top H\mathbf{a} - \lambda_+ N_+ + \lambda_- N_- + \frac{N}{2} \tag{6.28}$$

with constraint:

$$\mathbf{H}\mathbf{a} - \mathbf{c} + \lambda_+\mathbf{A}_+ + \lambda_-\mathbf{A}_-) = 0 \tag{6.29}$$

To remove the constraint, $\boldsymbol{\alpha}$ can be isolated in (6.29) to get:

$$\mathbf{a} = \mathbf{H}^{-1}(\mathbf{c} - (\lambda_+\mathbf{A}_+ + \lambda_-\mathbf{A}_-)) \tag{6.30}$$

Finally, this can be plugged into the dual problem to obtain a new dual formulation:

$$\begin{aligned}\max_{\lambda_+,\lambda_-} -\frac{1}{2}\begin{bmatrix}\lambda_+\\ \lambda_-\end{bmatrix}^\top \begin{bmatrix}\mathbf{A}_+^\top\mathbf{H}^{-1}\mathbf{A}_+ & \mathbf{A}_+^\top\mathbf{H}^{-1}\mathbf{A}_-\\ \mathbf{A}_-^\top\mathbf{H}^{-1}\mathbf{A}_+ & \mathbf{A}_-^\top\mathbf{H}^{-1}\mathbf{A}_-\end{bmatrix}\begin{bmatrix}\lambda_+\\ \lambda_-\end{bmatrix}^\top \\ + \begin{bmatrix}-N_+ + \mathbf{c}^\top H^{-1}\mathbf{A}_+\\ N_- + \mathbf{c}^\top\mathbf{H}^{-1}\mathbf{A}_-\end{bmatrix}^\top - \frac{1}{2}\mathbf{c}^\top\mathbf{H}^{-1}\mathbf{c} + \frac{N}{2}\end{aligned} \tag{6.31}$$

This problem avoids using any constraints, and can be solved analytically to produce optimal values for λ_+ and λ_-. By means of (6.30), $\mathbf{a}$, and thereby optimal $\boldsymbol{\alpha}$ and b, can be computed.

6.3.3 Step 2: Applying the trick

Finally, the trick behind the Sparse Greedy Approximation technique can be applied. The evaluation of these samples requires calculation of $\mathbf{H}^{-1}$ in the langrangian formulation of (6.31). As training is an iterative procedure, instead of recalculating $\mathbf{H}^{-1}$ at every iteration, we only need to perform a rank-1 update of $\mathbf{H}^{-1}$ when a new sample is found. Assuming a previous $\mathbf{H}_{old}^{-1}$ is known, we can calculate $\mathbf{H}^{-1}$ directly by:

$$\mathbf{H}_{new}^{-1} = \begin{bmatrix}\mathbf{H}_{old} & \mathbf{B}\\ \mathbf{B}^\top & C\end{bmatrix}^{-1} = \begin{bmatrix}\mathbf{H}_{old}^{-1} + (\mathbf{H}_{old}{}^{-1}\mathbf{B})^\top\gamma(\mathbf{H}_{old}^{-1}\mathbf{B}) & \gamma(\mathbf{H}_{old}^{-1}\mathbf{B})\\ -(\gamma(\mathbf{H}_{old}^{-1}\mathbf{B}))^\top & \gamma\end{bmatrix} \tag{6.32}$$

where $\gamma \equiv (C + \mathbf{B}^\top \mathbf{H}_{old}{}^{-1}\mathbf{B})^{-1}$. In our case, $\mathbf{B}$ is a vector with the new support vector and C is a constant. If the size $\mathbf{H}^{-1}$ is M, then this step can be performed in only $O(M^2)$. Given that M is typicaly much less than N, this results in a significant speed up.

6.3.4 Step 3: Selecting "good" support vectors

At each iteration in the algorithm a number of training samples Q must be selected (see 2a in table 6.1). The question is what samples to select? There are several options for selection criteria, such that a near-optimal solution is reached as soon as possible (see [108]). The one chosen here is simple, but have shown to perform well in practice ([82]). Among the Q samples, the sample is chosen which reduces the *dual infeasibility* the most, i.e. corresponds to the biggest (absolute) violation.

A probabilistic speed up

At each iteration of the sparse greedy algorithm, a number of samples Q are selected and evaluated at each iteration. Naturally, it is possible, but not practical, to select and search through all $(N - M)$ samples to find the optimal support vector. This would lead to a run time of $O(N^2 \cdot M^2)$. To speed up this step, we perform a second approximation. Instead of finding *the best* sample at each iteration it is sufficient to find a *good* sample.

The size of Q, also termed the quantile, can be based on some probabilistic theory from [108]. It is adequate to be pretty sure we will select at least one sample from the M best ones in the training set. If we want to find a sample from one of the M best ones with p probability from a training set of size N, the number of random samples we need to select is:

$$Q = \frac{\log(\frac{M}{N})}{\log(p)} \tag{6.33}$$

To give an example, if we have $\frac{M}{N} = 0.05$ and we want to select one of the top 5% with 95% probability, this equates to $Q = \frac{\log 0.05}{\log(0.95)} \approx 59$.

To give a more relevant example[7], if we have a training set with $N = 32000$ and want to select the 500 best ones ($M = 500$) with probability $p = \left(1 - \frac{500}{32000}\right) = 0.984$, we have $Q = \frac{\log\left(\frac{500}{32000}\right)}{\log(0.984)} = 264$.

[7] Why this example is relevant will be explained later in the KFD experiments section 7.1.2. However, a hint is that the size of the training set used in the example is $N = 32000$. This is the number of samples used for training each binary classifier in the large SVM and KFD experiments.

A higher quantile should be more likely to find good support vectors. However, as quantile affects training performance linearly, a compromise between speed and performance must be made.

6.3.5 Step 4: Selecting a termination criteria

The final problem to solve is finding a termination criteria such that the training stops when the solution is "optimal". There are several possible criteria available[8]. The ones found to be effective are:

Relative decrease in objective function value This criteria computes the *relative* improvement of the objective function for each iteration. This is independent of the actual objective function value, which can vary wildly from problem to problem. This could be set to e.g. 0.01, which means the training will terminate when the improvement at each iteration is less than 1%.

Leave One Out error At each (or every K) iterations, the leave-one-out error on the training set could be computed. When the error stops decreasing the training process could terminate. This criteria has been widely used in connection with SVMs and have shown to produce excellent results [57]. However, it has shown to be slightly unstable in some cases [59]. In addition, if the number of training samples are high, this step becomes fairly time consuming.

Validation set error Instead of measuring the accuracy on the training set, another popular and very successful criteria is to use the accuracy on a validation set. This was also used in the SVM experiments to select "good" hyperparameters. Here, it can be integrated into the training process and measure the validation performance of the model at each (or every K) iterations. When the prediction accuracy stops decreasing, the training process is terminated. In [59] and [27], this was shown to be the most stable measure of model performance, and consistently produced results very close to the test set accuracies.

Other termination criterias In addition to the above measures, two very simple and practical measures can be used in training. Since the training process can be very time consuming when M becomes high, a cap on the maximum number of support vectors can be used. Similarly, since the model is very unstable during the first few

[8] An examination of these are found in [59].

iterations, a cap on the minimum number of support vectors can also be added. These were found to be practical measures needed to produce good performing models.

A last, but important, measure is used. Early termination can happen when the algorithm is "unlucky" and selects "bad" support vectors. In practice, the decrease in objective function typically not completely monotonic, and therefore all measures are averaged (i.e. smoothed) over a window of fixed size.

6.3.6 Conclusion

The new approach consists of the four steps explained above. The resulting *Sparse Greedy KFD* reduces scaling to $O(N \cdot M^2 \cdot Q)$ and memory requirements to $O(N \cdot M^2)$, where N is the training set size, M the number of support vectors in the final solution, and Q the quantile. The claimed advantages of the approach [82] are many: If the required number of support vectors is low, this method completes training much faster than SVMs and with lower memory requirements, creating extremely sparse solutions with performance comparable to SVMs. In addition, because of the iterative approach, training can be performed "on-line". I.e. it enables easy updating of the model if new training data becomes available.

6.4 Classification

The created model contains support vectors as in the SVM solution. Thus classification can be done in exactly the same fashion as the SVM:

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b \tag{6.34}$$

where $sign(f(\mathbf{x}))$ is the predicted class label, and $f(\mathbf{x})$ an estimate of the class posterior probability.

6.5 Practical Issues

Several practical issues were addressed in order to take the step from theory to actual results. The sections below discusses the implementation of the sparse greedy KFD and also covers the issue of normalisation of data.

6.5.1 Implementation

A baseline C++ implementation of the Sparse Greedy KFD was gratefully received from Sebastian Mika. It uses a linear programming package (BLAS [65]) to solve the above problem. It is an efficient implementation, comprising of a single program that performs both training and classification. The program stored data in Matlab format, enabling easy analysis and post-processing of results.

To adapt the code to the problem in focus, several changes were made. Amongst the many modifications, the three main ones were:

1. It was impractical to have a program that *required* training before classification could be performed, since each model needs to evaluate several test sets. To allow more efficient use, the code was split up into a training and a classification module. This change resulted in heavy modification of the code, and also meant implementing a way to transfer the trained model including variables from the training module to the classifier.

2. In section 6.3.5, a termination criteria based on a validation set was suggested. As mentioned, this was found to be very successful in many cases [59, 27]. In addition, it was helpful to have a reliable quality measure of model performance during training. Therefore, an evaluation on a separate validation set was implemented into the training cycle. The measured accuracy was smoothed over a fixed size window to allow for more monotonous readings.

3. Finally, from both [82], [24] and from personal communications with Sebastian Mika, it was made clear that the threshold b obtained by the KFD method[9] was not always optimal. Therefore, a simple gradient descent search was implemented and used to find an improved threshold. Experiments after implementation confirmed the issue.

Numerous other implementation issues were addressed. However, most of these were minor problems of e.g. saving and loading the data, and properly logging the training and classification processes.

6.5.2 Data normalisation

As opposed to SVMs, where normalisation is required to ensure termination of the trainign process, the KFD can work with unnormalised data [23]. However, preliminary experiments showed that the performance improved when using normalised data

[9] See section 6.3.4.

in training. Moreover, to give equal "meaning" to the chosen hyperparameters for different binary problems all data was normalised[10].

6.6 Multi-class methods

One last issue must be considered before moving on to the experiments section: the mapping of the binary KFD classifiers to the multi-class case.

Although alternative multiclass schemes were available due to the KFD's ability to produce class posterior probability estimates [24], a proper comparison with the results found by SVMs were deemed more important. Therefore, the One-versus-One and DAGSVM classifers used in the SVM experiments were also employed in the KFD experiments.

6.7 Conclusion

The formulation of the Sparse Greedy Kernel Fisher Discriminant produces a very promising kernel machine. It claims to have a training time which scales $O(N^2)$ with the number of training samples N. Moreover, it has the ability to generate class probability estimates from classification output, it has a flexible training process, and finally, the created models in training typically have extremely few support vectors. These claims are very bold, and should naturally be verified on a practical problem. This is what will be done in the next chapter.

[10] If the data was left unnormalised, optimal values of the kernel parameter and C could risk being completely different on different binary problems. Normalisation made it easier to find "good" parameter sets when performing the rough exhaustive search for optimal paremeters.

Chapter 7

Experiments

In the following chapter, experiments using the KFD are described. First, the generalisation performance and the scaling behaviour will be examined in details on a limited number of binary classifiers. The second part will report results found from multi-class experiments comparable to the ones performed with SVMs in chapter 3.

7.1 Binary experiments

Several preliminary experiments were performed to evaluate the performance and verify the theoretical scaling properties of the KFD. In the experiments, a selection of 10 binary one-vs-one problems were randomly chosen from the 780 required for the full problem. For each binary problem, a 2000 frame training set was selected from the full training set with a separate validation set of 4000 frames; accuracy was measured on a 4000 frame test set. Training was stopped when a predefined maximum number of support vectors was reached. As in the SVM experiments, the Gaussian kernel is used. Finally, reported results are averaged over the 10 binary classifiers.

Several user-defined parameters are involved when training a KFD model. These need to be learned such that the discriminate performance of each classifier is optimal. The parameters are summarised in table 7.1. Additionally, the important factors in the resulting models are listed in table 7.2. How these are related to the user-defined parameters will be examined in this section[1].

[1]A note on the examination of the 4th important factor: classification time. As the resulting model of KFD training consists of a linear system exactly like a SVM model, a closer examination of classification speed was deemed unnecessary. However, it will be discussed later on in the multi-class experiments section.

Gaussian kernel parameter σ	The Gaussian kernel was chosen, since it was shown to be superior to the other kernels in the SVM experiments. The Gaussian kernel parameter σ needs to be chosen such that the classifier performs as well as possible.
Penalty term C	This parameter regulates the relation between the minimisation of the prior $P(\alpha) = \|\boldsymbol{\alpha}\|^2$ and minimising the classification error $\|\boldsymbol{\xi}\|^2$(in equation 6.20).
Quantile Q	The quantile Q defines the number of random training samples that are evaluated at each iteration. The value of Q must be high enough such that the samples added to the model (i.e. the support vectors) produce good quality solutions. However, it cannot be set too high, since Q increases training time linearly.
Amount of training data N	Generally, performance increases when more training data is included. Unfortunately, thetraining time increases when larger training sets are used.

Table 7.1: Overview of user-defined parameters in a binary KFD

7.1.1 Choosing σ and C

To examine the effect of σ and C on the created models, validation accuracies were found on models trained with different pairs of σ and C. Both parameters were plotted on a contour plot against the validation accuracies. The result can be found on figure 7.1. As seen, the area with optimal parameters are visible as a peak. However, notice how relatively flat the peaks are. This means, similarly to the SVM classifier, that a rough exhaustive search should be adequate to obtain well-performing parameters. Moreover, notice how σ affects the result much more than C does. Consequently, a more refined search for σ than C is a priority.

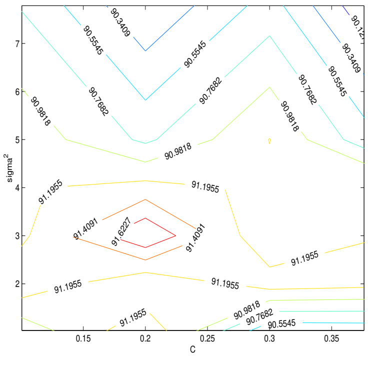

Figure 7.1: Contour plot of validation accuracy when varying C and σ^2 (sigma^2). Notice the fairly obvious peaks on the plot.

Number of support vectors (SVs)	The number of SVs in the resulting model defines the *sparsity* of the solution. This factor is implicitly chosen by the user through the parameters listed in table 7.1. Here, in contrast to the SVM solution, the user can define a limit on the number of SVs allowed in a solution.
Training time	Since time to train a model is significant, how the training time relate to the amount of training data used should be examined.
Performance on validation set	How the performance of the KFD compares with the performance of the SVM is very relevant, and should naturally be explored.
Classification time	Classification time is important if the classifier is to be used in a system where the run-time is crucial (as it is in a speech recognition system).
Quality of probability outputs	Since the KFD claims to enable direct interpretation of the classification outputs as class posterior probability estimates, the quality of the estimates and the distributions of the outputs should be examined.

Table 7.2: Important factors in a KFD model

7.1.2 Number of support vectors in solution

The number of support vectors in the solution M should affect the training time as $O(M^2)$. To verify this theoretical claim, the run-time was measured after each iteration and plotted against the number of iterations[2]. The result is shown on figure 7.2 (left). As can be seen, this relation scales approximately $O(m^2)$ as theory states.

Secondly, to examine how the quality of the solution increases with the increased number of support vectors, the Leave One Out error was computed at each iteration.

[2]As each iteration adds one support vector, this also equals the number of support vectors in the model.

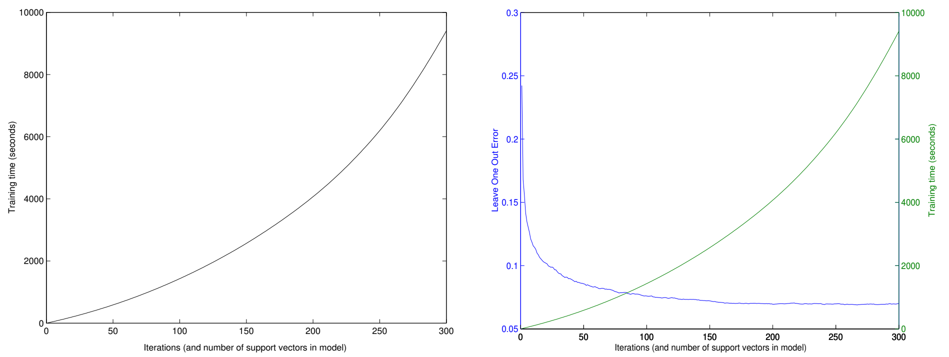

Figure 7.2: Left: Increase in training time when adding support vectors to the solution. Right: YY plot comparing training time vs. quality of solution using a quantile of 60. As seen, the convergence stops after approximately 200 iterations.

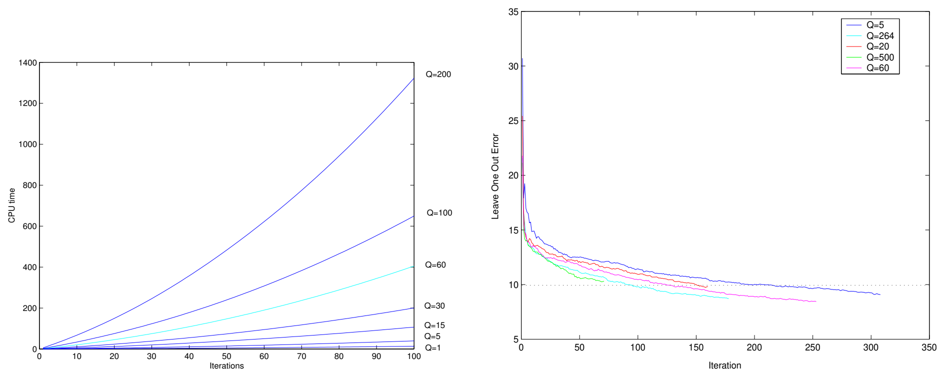

Figure 7.3: Left: Training time vs. number of iterations using different quantile sizes. Right: Speed of convergence when varying the quantile (number of). As seen, higher quantile results in faster convergence. Note that some runs were terminated earlier than others, as can clearly be seen on the figure. However, it still shows that the differences between the results are not extremely big.

The result is shown on figure 7.2 (right). Notice how convergence "flattens out" after approximately 200 iterations, resulting in a very sparse solution.

7.1.3 Varying the quantile size

To verify that training time scales linear with the size of the quantile, experiments with different quantile sizes were run. The averaged results can be seen on figure 7.3 (left). Looking at training times at the 100 iterations mark, the claim of linear training time scaling vs. size of quantile seems valid.

A larger quantile size examines more training patterns at each iteration. This should give it a higher chance of selecting "good" support vectors. To examine the effect of quantile size on the quality of the solution, the leave-one-out error was measured at each iteration. Experiments with different quantile sizes were run, and plotted on figure 7.3 (right). As expected, the figure shows that higher quantile sizes results in faster convergence.

In addition to faster convergence rates, it is noteable that a higher quantile also lead to a more monotonically increasing model performance. The reason is naturally that a higher quantile is more likely to find better training samples to add to the model at each iteration.

What quantile value has the best compromise between training speed and model quality? The theoretical "best" value was computed in section 6.3.4: If we estimate a model to have approximately 500 support vectors and the training set to have 32000 samples, then the support vectors should be chosen from the top $\frac{500}{32000} = 1.56\%$ of the

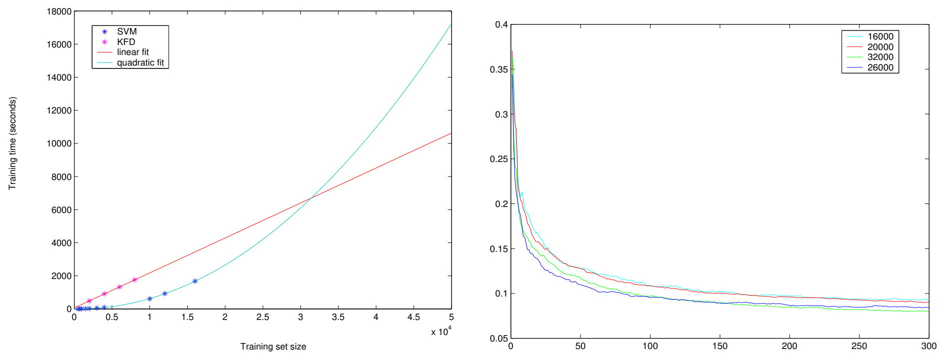

Figure 7.4: Left: Scaling of SVM and KFD. Horizontal axis is training set size and vertical axis is training time in seconds. It is clear that the scaling of the SVM is quadratic, whereas the scaling of the KFD is linear. Right: Leave One Out error convergence rate using different amounts of training data.

training patterns. If we will select these support vectors with $(1 - 0.0156) = 98.4\%$ likelihood, the "optimal" quantile should be: $Q = \frac{\log(500/32000)}{\log(1-500/32000)} \approx 264$. However, this quantile results in very slow training cycles, and is not practical. In addition, as seen on figure 7.3 (right) the differences in convergence rates are not big. Consequently, a higher number of iterations can make up for a lower quantile size. For example, using a quantile of 264 produces the same leave-one-out error after 140 iterations as using a quantile of 5 with 300 iterations. This is interesting, as the latter has a 10 times shorter training time. Based in this observation, a (lower) quantile of 60 was chosen. This was also the quantile size used in [108] and [82] where it produced excellent results.

7.1.4 Using different amounts of training data

A key problem with SVMs was the near-quadratic increase in training time when the training set size increased. One of the main advantages of the KFD method is the claimed *linear* scaling of training time vs. training set size. Naturally, this crucial claim had to be verified.

This was done by training models with different amounts of training data, while terminating the training when a certain number of support vectors was reached. The result can be seen on figure 7.4 (left), where it is compared with the scaling found by the SVM[3]. Based on the linear fit, the claim of linear scaling seems to be fairly accurate. Note that although the improvement in training time is not significant for smaller data sets, the linear scaling is extremely encouraging and a necessity for making kernel methods feasible on larger problems.

[3]Note: Absolute values for training time of the two methods should *not* be directly compared because they were implemented independently and were run on differing hardware.

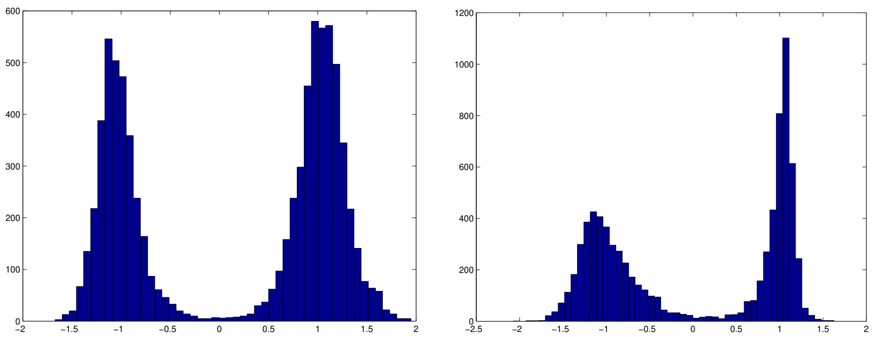

Figure 7.5: Histograms of outputs from the KFD classifier. Figure (a) shows two evenly distributed peaks at -1 and +1. Figure (b) shows output from another binary, where the two peaks are unevenly sized. However, clear peaks at +1 and -1 can still be seen.

To examine the effect on performance of adding more training data to the solution, the decrease in leave-one-out error during training was measured for different training set sizes. The result can be found on figure 7.4 (right). Although the performances are similar, it is obvious that more training data improves the solution. This conclusion was also reached in 3.1.3.

7.1.5 KFD vs. SVM

Comparing performance of the KFD and SVM on a limited number of binary problems can give an indication of the performance difference between the two classifiers. 10 binary models were built with KFD and SVMs using the Gaussian kernel. A rough parameter search was used to find the best performing parameters. Table 7.3 shows the results. As seen, the KFD produces almost the same performance as the SVM, but uses only 200 support vectors per classifier - a third of the SVM. Even with only 20 support vectors per KFD, performance remains surprisingly good.

Method	Validation accuracy
SVM	91.75%
KFD (Max 200 SVs)	91.50%
KFD (Max 20 SVs)	88.45%

Table 7.3: Binary mean validation accuracy on the 10 examined binary classifiers.

7.1.6 Class-conditional Posterior Probabilities

Finally, to verify the claim from section 6.2.6, that the resulting outputs can be estimated as class conditional posteriors, histogram plots of the validation set outputs of the 10 binaries were made. Two of the plots can be found on figure 7.5; one good and one bad example. As expected, due to constraints in equations (6.21) and (6.22), the outputs have the shape of Gaussian distributions clearly peaked around -1 and $+1$. In conclusion, the produced outputs from the KFD seem to be fairly well distributed, and should be useable as estimates of the class posterior probabilities.

7.1.7 Conclusion

From the above set of experiments, several observations can be made:

1. The performance of the KFD is close to the SVMs.
2. The claimed training speed of $O(N \cdot M^2 \cdot Q)$, where N is training set size, M number of support vectors in the final model, and Q the quantile size, holds true. If M and Q can be kept low and still retain the performance, the training time will scale near-linear with training set size.
3. The best quantile Q can be set lower than expected, and still produce good quality models.
4. Training time increases drastically with the number of support vectors. However, as KFD seems to create extremely sparse solutions with $M \ll N$, this does not slow down training much.
5. The claimed class-conditional posteriors can be found based on a predicted sample's distance from the decision boundary. This is an improvement over SVMs where this relation was difficult to obtain, even after using the post-fitting method described in chapter 4. This enables the KFD to be plugged into a full classification system in the same fashion as Neural Nets (see [93]).

7.2 Multi-class experiments

7.2.1 Introduction

In this section, results from multi-class KFD experiments are reported. Three multi-class schemes were examined: The One vs One classifier, the DAGSVM, and the weighted voting scheme. As these are the same schemes employed in the SVM experiments, it enables direct comparison between the KFD and SVMs.

As mentioned before, all three use the same training phase. This involves training one binary classifier for each combination of classes, resulting in $K(K-1)/2 = 780$ classifiers.

The section will first describe the experimental setup used in the experiments. Sequently, the results from the three schemes will be reported. Finally, conclusions will be drawn about the KFD method's capabilitites in phoneme recognition.

7.2.2 Experimental setup

The experimental setup used in the experiments was very similar to the one used in the SVM multi-class experiments. This was intentional to enable direct comparison with the SVM results from chapter 3. 16000 training patterns were used from each class[4], and a validation set of 4000 examples per class was used to find optimal parameters. Moreover, the Gaussian kernel was used and the quantile set to 60.

Each of the 780 binary classifiers were trained and evaluated. Since the classifiers could be trained independently, this step could be done in parallel to speed up the process. The machines used for training was 30 available Pentium-III 733Mhz Linux boxes, each with 128Mb memory[5].

7.2.3 Training

The training of each classifier started with normalisation of the data, and performing a rough exhaustive search to find "optimal" hyperparameters. After determining "reasonable" parameters choices, a proper training process was performed. Maximum allowed support vectors were set to 500, and the training process would terminate when the performance on the validation set stopped improving. The validation set accuracy was measured every 5 iterations and furthermore smoothed over a window of 5 readings. This was done to prevent early termination.

The resulting model and weights used for normalising the data were stored to be used in classification.

Comment on computational issues

Although the KFD improves upon the training and evaluation speed of SVMs, the time to train and evaluate 780 binary models are still a very comprehensive task. Moreover, as the software implementation was written for Linux machines this created further problems. Since the Linux processing power was limited compared to the Unix machines used by the SVM software, this caused the training cycles to take even longer

[4] As mentioned earlier, some classes have less than 16000 patterns. In these cases, all training samples will be used.

[5] This differs from SVMs, where experiments were run on faster Unix servers.

than the larger SVM experiments[6]. Due to this lack of fast and reliable processing power several compromises had to be made.

Firstly, the grid size of the exhaustive search was almost doubled, resulting in 3 times fewer parameter sets to search through. Although this still seemed to produce good quality parameter sets, this did affect performance.

Furthermore, a more practical problem was the limited amount of memory on the Linux machines. Since the program needed to store all training data N in memory and in addition used $O(M^2)$ memory during training, the 128Mb memory "barrier" was reached when $M \approx 250$. This slowed down training severely in the cases where the model needed more than 250 support vectors.

Finally, although the quantile of $Q = 60$ was enough to create good performing parameters, experiments in section 7.1.2 showed that convergece rates improved with higher a quantile. If more processing power or time had been available, a higher quantile could be used and this would lead to a more monotonically improving models, resulting in either a) even more sparse solutions, or b) better models.

7.2.4 One-vs-One classifier

The first multi-class scheme to be examined was the One vs. One scheme. To classify an unseen example, all 780 binary models are evaluated. For each model, a "vote" is given to the "winning" class[7]. The unseen example is classified as the class with most votes.

Method	MBA Val	MBA Test	Avg. SVs
SVM	96.5	**96.1**	2993.3
KFD	**96.7**	95.9	**229.2**

Table 7.4: Average binary results of the full KFD experiments. The results are listed against the best SVM results.

Binary results

The average binary accuracies on the validation and test sets are shown in table 7.4[8]. Here, the results from KFD and SVM experiments are compared.

[6] Moreover, the Linux machines were shared with fellow students, resulting in only 12 hours per day (on average). In addition, users had a habit of turning off the machines after use - effectively terminating the running jobs. This made processing power less reliable, and each experiment had to be completed using multiple training cycles, where each cycle consisted of retraining unfinished subjobs from the previous cycle.

[7] As mentioned earlier, a class "*wins*" the classification if the unseen example has been classified to lie on the classes' side of the decision boundary in equation (2.34)

[8] In the table, "MBA Val" and "MBA Test" are the mean binary accuracies on the validation and test set, respectively.

Several observations can be made:

- When looking at the results on the validation set, the binary mean of the KFD is 0.14% *better* than the binary mean of the SVM. This is extremely encouraging, since it shows that the approximate solution of the KFD can produce models equal or better to the computationally more expensive SVM.

- Even more interesting, look at the average number of support vectors in the solution. The SVM used 2993.3 support vectors and 26.9% of the total training data. In contrast, the KFD uses on average only 229.2 support vectors in each binary classifier, which is only 2.1% of the total amount of data used in training. Thus the KFD has effectively reduced the amount of needed support vectors by *13.9 times*. In other words, the KFD uses only 7.21% of the SVs needed for the SVM, *while still producing similar performance.* This reduction is important as it is linearly linked to the classification speed, resulting in a substantial 13.9 time reduction in classification time. An explanation for this pleasant observation can be found in the difference in the two kernel methods' discrimiative approaches. SVMs explicitly attempts to find the maximum margin between the classes and includes *all* training samples that are either misclassified or lies on the margin. If the overlap between the classes is high, the SVM will therefore include a lot of samples in the solution. In contrast, by looking at the *distribution* of the data, the KFD can model the data by using much fewer representative samples.

- However, all results using the KFD were not better than the SVM results. Looking at the very important binary test accuracy, the KFD is 0.18% *worse* than the SVM. In other words, the difference between the KFD validation set and test set accuracies is *double* that of the SVM (0.79% vs. 0.42%). Clearly, there is something wrong. How come the gap so big? The explanation could very likely be found in the way training is performed. The validation set accuracy is used as termination criteria in training, where it is measured every 5 iterations. Although this has been proven to be a excellent measure in several other experiments [41, 59], in this case the models slightly overfit the validation sets due to their close integration with the training process.

Method	MBA Val	MBA Test	Accuracy
SVM	96.5	**96.1**	**70.8%**
KFD	**96.7**	95.9	69.9%

Table 7.5: Overall classification rate of using the KFD in combination with the One vs One classification scheme. The relative increase in error rate from SVM to KFD is 3%.

Multi-class result

On table **??**, the result from the one vs one classification is shown. As seen, the overall accuracy of the KFD is 69.9%, which is 0.9% lower than the SVM performance. However, this was expected due to the lower binary test set accuracy reported in the previous section. The difference is not big, and it must be underlined that it is encouraging that such a sparse solution can produce almost the same performance.

Method	MBA Val	MBA Test	Accuracy
SVM	96.5	**96.1**	**71.4%**
KFD	**96.7**	95.9	70.2%

Table 7.6: Results of the DAGSVM experiment. The relative increase in error rate is 4%.

7.2.5 DAGSVM

To quickly summarise the DAGSVM classification scheme, it is a greedy decision-graph-based algorithm based on [53]. In this scheme a *sequence* of binary classifications are performed using the same one-vs-one binary classifiers as above. If the One vs One scheme was called “voting”, this scheme might be called a “knockout competition”. Only $(K-1)$ binary classifications are required to classify a test frame: $(K-1) = 39$ in our case. The result of the classifications are shown on table **??**.

Looking at the table, the overall prediction accuracy of 70.2% is lower than the one found in the SVM experiments. However, as with the One vs One scheme, it is still a decent result.

Interestingly, the difference between the SVM and KFD results (1.3%) in this experiment is slightly higher than it was in the One vs One results (only 0.9%). This may confirm a claim made in chapter 2 about the DAGSVM being less robust to errors that the One vs One scheme (see section 2.10.2). In other words, when the general quality of the individual binary classifiers are lower, it severely affects the overall classification performance.

7.2.6 Weighted voting

In chapter 4, a multiclass scheme was described that took into account the class posterior probabilities estimates produced in classification. To briefly summarise, it is a variant of the One vs One scheme, which weight the ”votes” such that each vote assigned to the classes are ”weighted” with the class posterior probability. For example, if the binary classifier of class A vs class B predicts that an unseen example is 40% likely to belong to class A and 60% likely to belong to class B, then class A receives a weighted vote with value 0.4 and class B receives a vote with value 0.6. This way, a more refined classification can be made. Unfortunately, in chapter 4 it was found a few influential classes ruined the overall prediction quality.

As KFD classification can directly be interpreted as posterior probability estimates, it was interesting to see if it could improve upon the results found by the SVM in chapter 4. The weighted voting scheme were carried out and results can be seen on table **??**.

Method	MBA Val	MBA Test	Accuracy
SVM	96.5	**96.1**	54.3%
KFD	**96.7**	95.9	**59.1%**

Table 7.7: Results from the weighted voting experiment.

Looking at the table, it is encouraging to see that the KFD performs better than the SVM. Although the influential classes still causes the result to be much less than the DAGSVM and One vs One schemes, the substantial improvement over the SVM result by 4.8% (11estimates are better than the SVM's sigmoid-fitted estimates. This confirms the conclusion made in [8] about the KFD's ability to produce quality class-conditional posterior estimates.

7.3 Conclusion

7.4 Conclusion

The above experiments show that the Kernel Fisher Discriminant kernel machine is able to create capable framewise phoneme classifiers that can generalise well even in very difficult situations. By looking at the *distribution* of data instead of the margin, the KFD offers solutions to many of the problems of the SVM. First of all, it took a big step in the right direction in terms of scalability to larger data sets. The training time increased near-linearly with the amount of training data, which is a big improvement upon the scaling of the SVM. As noted earlier, this is a key to solving the "typical" speech problems which are much larger than the TIMIT data set[9].

Secondly, it enabled direct interpretation of outputs as probability estimates, which was shown to be better estimates than the post-fitting method applied to the SVM. Thirdly, the method created extremely sparse solutions, using only a fraction of the support vectors of the SVM. This reduced classification time by a drastic 13.9 times. Finally, the iterative training method allows for flexible on-line updating of models.

[9] Such as the Switchboard data set, which as over 200 times more data [37].

The overall results of the two best classifiers were 69.9% for the one-vs-one classifier and 70.2% for the DAGSVM scheme. Compared with the best found result of the SVM, 71.4%, these results are slightly worse. However, conditions under which the KFD experiments were run might partally be a reason for the worse result[10].

Moreover, although the weighted voting scheme's result of 59.1% is significantly worse than the other two results, it improved by 4.8% upon the SVM result. This indicates that the KFD is able to produce quality probabilistic estimates from classifications.

Still, the method was not without it's problems. Introducing a new user-defined parameter, the *quantile* Q, required even further tuning of parameters. In addition, if the number of support vectors are high or if a large quantile is needed to produce good solutions, the training time of the KFD can easily be even higher than that of the SVM. It was only when solutions were sparse and when a low quantile was adequate to produce good quality solutions that the KFD outperformed the SVM. Fortunately, that seemed to be the case in most experiments.

As a final note, a concluding experiment was proposed but had to be cancelled due to lack of time. A closer examination into the individual binary classifiers from both the SVM and KFD systems revealed a fairly interesting fact: The difference in model performances between the two systems varied a lot. The standard deviation on the *difference* in validation set accuracies was a surprising 1.1%. In other words on some binary problems the SVM performed much better than the KFD, and vice versa. Hence, one should be able to take the best binary classifiers from both systems to create a combined classifier with very high performance[11].

[10] That the compromises taken during training did affect performance was proven by looking at the generalisation performance of the 10 binary classifiers examined in section 7.1. Examinations concluded that a more thorough parameter search, a higher quantile and more support vectors lead to an average improvement of about 0.4% compared to the results found by the exhaustive parameter search in multi-class training.

[11] The average validation set accuracy of such a system is 97.1%, which is an improvement upon the SVM system of 0.54%. With such a high increase, the overall classification accuracy should be expected to improve significantly.

End of Part II

Chapter 8

Conclusion and future work

The goal of this thesis was to examine the feasibility of applying kernel machines to the problem of frame classification. This was done, and the Support Vector Machines and Kernel Fisher Discriminant were found to create a capable learning machines with good generalisation abilities. Even with the tremendous amount of overlap amongst classes, they could successfully model the speech data and produce liable classifications.

In the first part, the SVM method was examined. Through maximising the margin of separability between classes and thereby implementing the Structural Risk Minimisation principle, it demonstrated that it could counteract the *curse of dimensionality* and generalise well even when the data contained numerous irrelevant features.

In chapter 3 I described two different schemes for extending the inherently binary SVM to the multi-class domain. The results show that the schemes can effectively retain the generalisation abilities of the binary SVM. The best reported result of 71.4% is competitive with the best results found in literature, even under sub-optimal conditions where only 40% of the training data was used. The experiments showed that employing the Gaussian kernel lead to better generalisation performance than by using the linear or polynomial kernels. The reason was believed to be the kernel's potential to map the data in input-space into an infinite dimensional space. Furthermore, the attempts to include additional information in the feature vectors revealed a substantial amount of information about each frame to be hidden in the neighbouring frames. Finally, it was made clear that the a key to better generalisation performance is the amount of training data used.

In chapter 4 a method was described for retrieving posterior probability estimates from the distance-based outputs of the SVMs. The estimates were shown to be acceptable, although only approximate, estimates of the actual posterior probabilities. The extension enables SVM classifiers to be used as discriminative front-end classifiers in a full speech recognition system.

In the second part of the thesis, alternative kernel machines were reviewed. These were proposed as means to solve some of the many problems found in the SVM method. A selected method, the Kernel Fisher Discrimiant, was introduced. By looking at the *distribution* of the data, the classifier were able to separate classes with high overlap, and could be linked to a Bayesian framework which allowed a probabilitic interpretation of classification outputs. In addition, it was shown to create extremely sparse solutions using a very flexible training process. Lastly, and most importantly, the KFD solved the key problem of the SVM: the severe increase in training time when using large data sets. The KFD was shown to have training time that *scaled linear with the training set size.* Experiments in chapter 7 verified these claims, and showed that the KFD had near-similar generalisation performance to the SVM. This was achieved under less optimal conditions while using 14 times fewer components in the solution compared to the SVM.

However, although the KFD proved to be a big step in the right direction, all hurdles in making kernel machines a viable options in real-time speech recognisers were not solved. Firstly, multi-class training is still a computationally *very* expensive problem to solve, requirering fast and reliable processing power. Secondly, the number of binary classifiers to process still presents a logistically difficult problem. It is not easy to keep track, and fully optimise, hundreds of binary classifiers.

8.1 Future work

The following are some thoughts on what could be done to improve the methods further.

8.1.1 Improving classification speed

Even though the focus of the thesis was *not* to think in terms of speed and applicability to real-time speech recognition systems, it must not be forgotten. By creating sparse solutions, the Kernel Fisher Discriminant drastically reduced the classification time compared to the SVM. However, classification still took far more time than what is allowed in a real-time system. That is, it took much longer time to classify all samples than the actual length of the speech[1]. Fortunately, there are several obvious ways to improve this.

[1] Classification times were roughly 120x real-time speed for the One-vs-One multi-class scheme and 6x real-time speed for DAGSVM scheme. This was recorded on Pentium 3-733MHz Linux machines with 128Mb of memory.

1. *Reducing the amount of binary SVMs to evaluate.* Even though the DAGSVM algorithm drastically reduced the number of binary SVMs to evaluate compared with the One vs. One classifier, it still had great redundancy in classification since *all* 40 phoneme classes were considered. A pruning technique could reduce this number by exploiting the phonological groupings of the phonemes[2].

2. *Reducing number of features.* Currently, the best results are found with samples containing 195 features each. However, since these are derived from the basic 12 Mel Cepstrum coefficients + energy, one could consider the remaining 182 features as "extra information". It should easily be possible to reduce this number significantly without a notable performance hit - especially since this reduction could be done individually for each binary classifier. The reduction could be done by e.g. performing a simple Principal Component Analysis (PCA) and remove features with low eigenvalues.

3. *Reducing number of classifications.* As a last, although less ideal, option, the number of frame-by-frame classifications could be reduced. In the current setup, there are 100 frame classifications per second. Since each phoneme lasts 65ms on average [43], there are a high redundancy in the number of classifications per phoneme. Instead of classifying each frame, the kernel-based classifier system could be used as an "enquiry" system: At any given time during classification, the predictions of previous frames are known, including the confidence in the predictions. Using these as a "guideline", evaluation of the next unknown frame could simply test whether or not the new frame is from the same phoneme class as the previous prediction. If the prediction is similar, then a "full" classification process is not needed. This pruning technique could potentially reduce computations by another significant amount. However, this technique is dangerous to use as it could cause a high number of *deletions* (i.e. fail to detect some phonemes [58]).

The above ideas show that there are still plenty of redundancy in classification, and that the 0.5x real-time typically assigned to a front-end ([93]) should be possible.

8.1.2 Improving performance

Both examined kernel machines have shown to produce high quality classifiers. However, there are three very simple ways to further improve performance.

Firstly, as described in both the SVM and KFD experiment chapters, the multi-class experiments were performed under sub-optimal conditions. The simple parameter

[2] For example, separating vowel and consonant phonemes.

searches were *very* rough, and a better tuning of parameters would most likely lead to a decent increase in performance.

Secondly, in the case of the KFD, another very straightforward way of obtaining better performance is to simply increase the quantile Q. In section 7.1.2, this was shown to produce faster, more monotonous, convergence, and thus better performance.

Finally, perhaps the most simple way of boosting performance is to train the classifiers on the *full* TIMIT training set. In the most extensive experiments only 40% of the data was used. A 250% increase in training set size could potentially improve results even further.

In addition to these three improvements, several less straightforward measures were considered. The following attempts to summarise the ideas.

- *Improving the existing multi-class scheme.* The two classification schemes examined in the experiments consisted of 780 *independent* binary classifiers. In other words, there were no modelling of inter-dependencies among the models. Methods to learn such connections could very likely improve the binary to multiclass mapping.

- *Using alternative multi-class schemes.* The extensive number of independent binary classifiers in the pair-wise multi-class schemes is difficult to handle. Some alternative schemes have been proposed that reduce this number, such as Error Correcting Output Codes [25, 22] and a system based on *binary phonological features* in the language [61].

- Using *alternative kernels.*
  In the experiments, examination of different kernels were limited to three very popular ones. However, other kernels exist that could possible help produce even better classifiers. Among the interesting kernels considered were data-dependent kernels [54], splines [105], automatic relevance detection kernels [78], and non-parametric kernels [51]. Additionally, recurrent kernels are also a possibility [118]. The principle of feeding back predictions into the learning machine has been used with success in Recurrent Neural Networks, the idea might be used here to gain similar improvements.

- *Improving KFD implementation.*
  The sparse greedy approximation technique used in KFD training offers a very flexible framework. Due to it's iterative nature, existing models can easily be retrained with new data, to add more support vectors to the solution. There are several obvious ways to improve the implementation even further. These involve a) make training faster and b) to improve the quality of the resulting solutions.

Firstly, to improve training time, caching can be used for the evaluated samples. This was implemented in the SVM software used in chapter 3, and improved training time significantly (see [56] for details).
Secondly, to improve quality of the models, consider what is done at each iteration during training. The algorithm searches for a "good" training sample, which is subsequently *added* to the solution. How about allowing more operations that simply "adding" samples to the solution? In [86] other operations were proposed. Among them were an "*exchange*" operation: If a newly chosen sample S_{new} was found to be very similar to a support vector S_{old} from the existing solution, then S_{old} was replaced by S_{new} if the value of objective function decreased by using S_{new} instead of S_{old}. By doing this, the solution could remain sparse while still improving performance.

The above list attempted to explain some of the possible future improvements. However, the list is by no means exhaustive; plenty of other proposals were found during the search for improved methods[3], and to explain them all could easily use up another hundred pages. Nonetheless, kernel machines remains an extensively researched area, and it is almost certain that even better machines will be discovered in the future.

8.2 Final words

The question still remains: How far are kernel-machines from being used in a practical speech recognition system? As concluded, the Kernel Fisher Discriminant proved to be a big step in the right direction compared to the popular Support Vector Machines, solving many of the key hurdles. However, to be realistic, there's still far to go. Taking a brief look at the above list makes it obvious that a lot more potential work must be done before the problem is solved.

The task of this thesis was frame-by-frame classification. It is important to underline that this is only one part of many in a full speech recognition system. Future work should broaden the task to include *phoneme recognition*. In other words, decoding the framewise outputs to recognise *whole* phonemes. If this could be done, kernel machines could be used in a similar fashion to Neural Networks in state-of-the-art speech recognition systems such as [93]. So even though there is still far to go, the possibility is within reach.

[3] Some more obscure than others.

Appendix A

Kernel Requirements

The following theorems are taken from [11].

A.1 Symmetry Condition

Let $K(x, y)$ be a real symmetric function on a finite input space, then it is a kernel function if and only if the matrix K with components $K(x_i, x_j)$ is positive semi-definite.

A.2 Mercer's Theorem

This theorem must be satisfied by a functional for a pair Φ, H to exist.

For a compact subset, $C \in \Re^N$, we have:

If $K(x, y)$ is a continuous symmetric kernel of a positive integral operator T, i.e.

$$(Tf)(\mathbf{y}) = \int\limits_C K(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) d\mathbf{x} \tag{A.1}$$

with:

$$\iint\limits_{C \times C} K(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) f(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0 \tag{A.2}$$

For all $f \in C \times C$ then it can be expanded in a uniformly convergent series in the eigenfunctions $\Phi_j$and positive eigenvalues λ_j of T, thus:

$$K(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{N} \lambda_j \Phi_j(\mathbf{x}) \Phi_j(\mathbf{y}) \tag{A.3}$$

where N_e is the number of positive eigenvalues.

This theorem holds for general compact spaces, and generalises the requirements to infinite feature spaces. Equation (A.2) generalises the semi-positivity condition for finite spaces given in the theorem in section A.1. The expansion in (A.3) is a generalisation of the usual concept of an inner product in Reproducing Hilbert Spaces with each dimension rescaled by $\sqrt{\lambda_j}$.

Note that for specific cases, it may not be easy to check whether Mercer's condition is satisfied. Equation (A.3) must hold for every function f with finite L2-norm (i.e. which satisfies (A.3)).

Appendix B

Phonemes in the TIMIT speech database

On table B.1, an overview of the TIMIT speech corpus can be found.

Phoneme	Training set			Test set	Frequency
	Total	Training	Validation		
/aa/	27339	16649	10690	10698	0.026
/ae/	30843	18697	12146	10470	0.025
/ah/	37788	22654	15134	14652	0.036
/ao/	22703	13974	8729	9255	0.023
/aw/	11636	7303	4333	3751	0.009
/ay/	29526	17763	11763	10975	0.027
/b/	15665	9357	6308	6691	0.016
/ch/	6972	4131	2841	2244	0.005
/d/	25462	15780	9682	8519	0.021
/dh/	13729	8332	5397	5160	0.013
/eh/	30173	17853	12320	11264	0.027
/er/	38674	24152	14522	16335	0.040
/ey/	28480	17406	11074	10611	0.026
/f/	22509	13737	8772	9408	0.023
/g/	10183	6250	3933	3985	0.010
/hh/	11038	6913	4125	3831	0.009
/ih/	70017	42160	27857	24259	0.059
/iy/	43533	26278	17255	16736	0.041
/jh/	6133	3727	2406	1877	0.005
/k/	43997	26840	17157	14386	0.035
/l/	35245	21292	13953	14673	0.035
/m/	22996	13750	9246	9092	0.022
/n/	40590	24294	16296	14339	0.035
/ng/	7344	4390	2954	2461	0.006
/ow/	20958	12910	8048	7858	0.019
/oy/	4981	3066	1915	2241	0.005
/p/	29482	17595	11887	10921	0.027
/r/	25998	15769	10229	10805	0.026
/s/	68871	41672	27199	24874	0.061
/sh/	15285	9410	5875	5583	0.014
/sil/	172030	104019	68011	63214	0.154
/t/	48625	29846	18779	16829	0.041
/th/	6725	3919	2806	2258	0.005
/uh/	3771	2226	1545	1655	0.004
/uw/	19345	11648	7697	5561	0.014
/v/	11953	7419	4534	4229	0.010
/w/	13263	7979	5284	5825	0.014
/y/	5347	3274	2073	2118	0.005
/z/	30567	18591	11976	10641	0.026
/zh/	1204	697	507	636	0.002
Total	1110980	673722	437258	410920	1.000

Table B.1: Overview of phonemes in the TIMIT database. A subset of the full training set is used as a separate validation set. The last column shows the frequencies of the test set. Note that /sil/ represents silence.

Appendix C

Paper submitted to ICSLP 2002

The following paper is based on parts of the work done for this thesis. It has been submitted to the International Conference for Spoken Language Processing, in Denver Colorado Sept 16-20 2002.

Page 1 of paper

Page 2 of paper

Page 3 of paper

Page 4 of paper

Bibliography

[1] V. G. T. at al. Bayesian inference for ls-svms on large data sets using the nystrom method. Technical report, ESAT-SISTA, K.U.Leuven (Leuven, Belgium), 2001.

[2] R. S. Att. The boosting approach to machine learning an overview.

[3] H. B., S. J.A.K., and D. M. B. A comparison of iterative methods for least squares support vector machine classifiers. Technical report, ESAT-SISTA, K.U.Leuven (Leuven, Belgium), 2001.

[4] P. Bartlett and J. Shawe-Taylor. Generalization performance of support vector machines and other pattern classifiers. In *Advances in Kernel Methods — Support Vector Learning*, pages 43–54. MIT Press, 1999.

[5] G. Baudat and F. Anouar. Generalized discriminant analysis using a kernel approach. *Neural Computation*, 12(10):2385–2404, 2000.

[6] F. Beaufays, H. Bourlard, H. Franco, and et al. Neural networks in automatic speech recognition.

[7] K. Bennett and J. Blue. A support vector machine approach to decision trees, 1997.

[8] B. Bradshaw, B. Scholkopf, and J. C. Platt. Kernel methods for extracting local image semantics. Technical report, Microsoft Research, Redmond, USA, 2001.

[9] J. Bridle. *Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationship to Statistical Pattern Recognition.* Springer-Verlag, 1989.

[10] C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.

[11] C. Campbell. An introduction to kernel methods. In R. Howlett and L. Jain, editors, *Radial Basis Function Networks: Design and Applications*, page 31. Springer Verlag, Berlin, 2000.

[12] Chen and Hanzo. Block-adaptive kernel-based cdma multiuser detector.

[13] R. Chen and L. Jamieson. Experiments on the implementation of recurrent neural networks for speech phone recognition, purdue university, west lafayette, us, 1998.

[14] S. Chen, S. Gunn, and C. Harris. The relevance vector machine technique for channel equalization application.

[15] K. K. Chin. Support vector machines applied to speech pattern classification. Master's thesis, Engineering Department, Cambridge University, 1999.

[16] R. Chun. A hierarchical feature represenation for phonetic classification. Master's thesis, MIT, Cambridge, 1996.

[17] P. Clarkson and P. Moreno. On the use of support vector machines for phonetic classification. In *Acoustics, Speech and Signal Processing, volume II*, pages 585–588, 2000.

[18] R. Cole. Alphadigit corpus, 1997.

[19] R. Collobert and S. Bengio. Support vector machines for large-scale regression problems. *IDIAP*, 2000.

[20] R. Collobert and S. Bengio. Support vector machines for large-scale regression problems. *Machine Learning Research*, 1:143–160, 2001.

[21] R. Cooley. Classification of news stories using support vector machines. In *Proc. 16th International Joint Conference on Artificial Intelligence Text Mining Workshop*, 1999.

[22] K. Crammer and Y. Singer. On the learnability and design of output codes for multiclass problems. In *Computational Learing Theory*, pages 35–46, 2000.

[23] D. DeCoste and ichael C. Burl. Support vector machines and kernel fisher discriminants: A case study using electronic nose data, 2001.

[24] V. G. T. et al. A bayesian framework for least squares support vector machine classifiers. Technical report, ESAT-SISTA, K.U.Leuven (Leuven, Belgium), 2000.

[25] V. G. T. et al. Multiclass ls-svms: Moderated outputs and coding-decoding schemes. Technical report, ESAT-SISTA, K.U.Leuven (Leuven, Belgium), 2000.

[26] T. Evgeniou, L. Perez-Breva, M. Pontil, and T. Poggio. Bounds on the generalization performance of kernel machine ensembles. In *Proc. 17th International Conf. on Machine Learning*, pages 271–278. Morgan Kaufmann, San Francisco, CA, 2000.

[27] T. Evgeniou, L. Perez-Breva, M. Pontil, and T. Poggio. Bounds on the generalization performance of kernel machines ensembles. In *International Conference on Machine Learning*, 2000.

[28] R. A. Fisher. The use of multiple measurements in taxonomic problems. In *Ann. Eugenics*, pages 111–132, 1936.

[29] J. Friedman. Another approach to polychotomous classification. Technical report, Stanford University, UA, 1996.

[30] G. Fung and O. L. Mangasarian. Proximal support vector classifiers. *KDD-2001*, pages 77–86, 2001.

[31] G. Fung, O. L. Mangasarian, and A. J. Smola. Minimal kernel classifiers.

[32] A. Ganapathiraju. *Support Vector Machines for Speech Recognition*. PhD thesis, Mississippi State University, US, 2002.

[33] A. Ganapathiraju and J. Picone. Hybrid svm/hmm architectures for speech recognition. In *Neural Information Processing Systems*, 2000.

[34] J. Garafolo. Getting started with the darpa timit cd-rom: An acoustic phonetic speech database, 1988.

[35] T. D. K. E. Gates and A. Masters. Exact simplification of support vector solutions. In *Journal of Machine Learning Research 2*, pages 293–297, 2001.

[36] M. G. Genton. Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, 2:299–312, 2001.

[37] J. J. Godfrey and E. Holliman. Switchboard-1 release 2.

[38] T. Graepel and R. Herbrich. The kernel gibbs sampler. In *NIPS*, pages 514–520, 2000.

[39] Y. Guermeur. Combining discriminant models with new multi-class svms. Technical report, LORIA INRIA-Lorraine, Vandaeuvre-les-Nancy cedex, France, 2000.

[40] S. Gunn. Support vector machines for classification and regression. Technical report, University of Southampton Image Speech Intelligent Systsmes Group, 1997.

[41] I. Guyon, B. Boser, and V. Vapnik. Automatic capacity tuning of very large VC-dimension classifiers. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5, pages 147–155. Morgan Kaufmann, San Mateo, CA, 1993.

[42] I. Guyon and D. Stork. Linear discriminant and support vector classifiers. In *Advances in Large Margin Classifiers*, pages 147–169. MIT Press, 2000.

[43] S. v. V. H. Yang and H. Hermansky. Relevancy of time-frequency features for phonetic classification measured by mutual information. In *ICASSP 99*, 1999.

[44] J. E. Hamaker. Sparse bayesian methods for continuous speech recognition, phd thesis proposal, 2002.

[45] R. Herbrich. The structure of version space.

[46] R. Herbrich and T. Graepel. Large scale bayes point machines. In *NIPS*, pages 528–534, 2000.

[47] R. Herbrich, T. Graepel, and C. Campbell. Bayesian learning in reproducing kernel hilbert spaces – the usefulness of the bayes point, 1999.

[48] R. Herbrich, T. Graepel, and C. Campbell. Robust bayes point machines, 2000.

[49] R. Herbrich, T. Graepel, and C. Campbell. Bayes point machines. *Journal of Machine Learning Research*, 1:245–279, 2001.

[50] C. Hsu and C. Lin. A comparison on methods for multi-class support vector machines. Technical report, National Taiwan University, Taiwan, 2001.

[51] A. F. Institute. Adaptive sparseness using jeffreys prior.

[52] M. F. Instituto. Bayesian learning of sparse classifiers.

[53] N. C. J. Platt and J. Shawe-Taylor. Large magin dags for multiclass classification. Technical report, Microsoft Research, Redmond, US, 1999.

[54] T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *Advances in Neural Information Processing Systems 11*, 1999.

[55] T. Joachims. Text categorization with support vector machines: learning with many relevant features. In *Proc. 10th European Conference on Machine Learning ECML-98*, pages 137–142, 1998.

[56] T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

[57] T. Joachims. Estimating the generalization performance of an SVM efficiently. In *Proc. 17th International Conf. on Machine Learning*, pages 431–438. Morgan Kaufmann, San Francisco, CA, 2000.

[58] D. Jurafsky and J. H. Martin. *Speech and Laguage Processing*. Prentice-Hall, 2000.

[59] S. Keerthi. Efficient tuning of svm hyperparameters using radius margin bounds and iterative algorithms, 2001.

[60] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. A fast iterative nearest point algorithm for support vector machine classifier design. *IEEE-NN*, 11(1):124, January 2000.

[61] S. King and P. Taylor. Detection of phonological features in continious speech using neural networks. In *Computer Speech and Laguage*, pages 333–353, 2000.

[62] Y. S. Koby Crammer. On the algorithmic implementation of multiclass kernel-based vector machines. In *Journal of Machine Learning Research 2*, pages 265–292, 2001.

[63] J. T.-Y. Kwok. Moderating the outputs of support vector machine classifiers. *IEEE-NN*, 10(5):1018, September 1999.

[64] N. D. Lawrence and B. Schölkopf. Estimating a kernel fisher discriminant in the presence of label noise, 2001.

[65] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage.

[66] Y.-J. Lee and O. L. Mangasarian. Rsvm: Reduced support vector machines. In *In Proceedings of the First SIAM International Conference on Data Mining*, 2001.

[67] Y. Li and P. M. Long. The relaxed online maximum margin algorithm. *Machine Learning*, 46(1/3):361, 2002.

[68] C. Lin. the convergence of the decomposition method for support vector machines, 2000.

[69] Y. Lin, G. Wahba, H. Zhang, and Y. Lee. Statistical properties and adaptive tuning of support vector machines. Department of statistics technical report, University of Wisconsin-Madison, 2000.

[70] S. Lukas and L. Vandewalle. Sparse least squares support vector machine classiers, 2000.

[71] E. M. B. M. A. Aizermann and L. I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. In *Automation and Remote Control 25*, pages 821–837, 1964.

[72] C. Ma and M. A. Randolph. A support vector machine-based rejection technique for speech recognition. In *ICSLP, Beijing, China*, 2000.

[73] D. MacKay. Introduction to gaussian processes, 1997.

[74] M. Malayath. Data-driven methods for extracting features from speech. Technical report, Indian Institure of Technology, Madras, India, 2000.

[75] O. Mangasarian and D. R. Musicant. LSVM Software: active set support vector machine classification software, 2000. www.cs.wisc.edu/∼musicant/lsvm/.

[76] J. Marques and P. J. Moreno. A study of musical instrument classification using gaussian mixture models and support vector machines. Technical report, Cambridge, US, 1999.

[77] E. Mayoraz and E. Alpaydin. Support vector machines for multiclass classification. In *International Workshop on Artificial Neural Networks (IWANN99)*, 1999.

[78] M. McPhail. Support vector machines for time-series prediction. Technical report, University of Edinburgh, 2001.

[79] S. Mika, G. Ratsch, and K.-R. Muller. A mathematical programming approach to the kernel fisher algorithm. In *NIPS*, pages 591–597, 2000.

[80] S. Mika, G. Rätsch, J. Weston, B. Schölkopf, and K.-R. Müller. Fisher discriminant analysis with kernels. In Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, *Neural Networks for Signal Processing IX*, pages 41–48. IEEE, 1999.

[81] S. Mika, G. Ratsch, J. Weston, B. Scholkopf, A. Smola, and K. Muller. Invariant feature extraction and classification in kernel spaces, 2000.

[82] S. Mika, A. Smola, and B. Scholkopf. An improved training algorithm for kernel fisher discriminants, 2001.

[83] P. Moreno and R. Rifkin. Using the fisher kernel method for web audio classification. In *ICASSP*, Istanbul, Turkey, 2000.

[84] K.-R. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik. Predicting time series with support vector machines. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 243–254. MIT Press, 1999.

[85] M. G. N. Smith and M. Niranjan. Data-dependent kernels in svm classification of speech patterns. Technical report, Cambridge University, UK, 2001.

[86] P. B. Nair, A. Choudhury, and A. J. Keane. Some greedy algorithms for sparse nonlinear regression, 2001.

[87] B. K. Natarajan. Sparse approximate solutions to linear system. *SIAM Journal on Computing*, 24(2):227–234, April 1995.

[88] E. Osuna, R. Freund, and F. Girosi. An improved training algorithm for support vector machines. In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII — Proceedings of the 1997 IEEE Workshop*, pages 276 – 285, New York, 1997. IEEE.

[89] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *Proceedings of CVPR'97, Puerto Rico*, 1997.

[90] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 185–208, Cambridge, MA, 1999. MIT Press.

[91] J. Platt. Probabilities for SV machines. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–74, Cambridge, MA, 2000. MIT Press.

[92] L. R. Rabiner and B. H. Juang. *Fundamentals of Speech Recognition.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1993.

[93] A. Robinson, G. Cook, D. Ellis, E. Fosler-Lussier, S. Renals, and D. Williams. Connectionist speech recognition of broadcast news. In *Speech Communication*, 2001.

[94] T. Robinson. An application of recurrent nets to phone probability estimation. In *IEEE Transactions on Neural Networks, vol. 5, no. 2*, pages 298–305, 1994.

[95] V. Roth and V. Steinhage. Nonlinear discriminant analysis using kernel functions technical report nr, 1999.

[96] J. Salomon. Support vector machines for phoneme classification, 2001.

[97] B. Sch, o Smola, and K. uller. Kernel principal component analysis.

[98] M. Schmidt and H. Gish. Speaker identification via support vector classifiers. In *ICASSP, volume 1*, 1996.

[99] B. Schölkopf. *Support Vector Learning.* R. Oldenbourg Verlag, Munich, 1997. To order click here or here.

[100] B. Schölkopf, C. Burges, and A. Smola. Introduction to support vector learning. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, pages 1–22. MIT Press, 1998.

[101] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings, First International Conference on Knowledge Discovery & Data Mining*, Menlo Park, 1995. AAAI Press.

[102] B. Schölkopf, S. Mika, C. J. C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. J. Smola. Input space vs. feature space in kernel-based methods.

[103] A. Schwaighofer and V. Tresp. The bayesian committee support vector machine. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Artificial Neural Networks - Proceedings of ICANN 2001*, pages 411–417. Springer Verlag, 2001.

[104] S. K. Shevade, S. S. Keerthi, C. Bhattacharyya, and K. R. K. Murthy. Improvements to the SMO algorithm for SVM regression. *IEEE-NN*, 11(5):1188, September 2000.

[105] A. Smola. Regression estimation with support vector learning machines, 1996.

[106] A. J. Smola. Regression estimation with support vector learning machines. Master's thesis, Technische Universität München, 1996.

[107] A. J. Smola and P. L. Bartlett. Sparse greedy gaussian process regression. In *NIPS*, pages 619–625, 2000.

[108] A. J. Smola and B. Schölkopf. Sparse greedy matrix approximation for machine learning. In *Proc. 17th International Conf. on Machine Learning*, pages 911–918. Morgan Kaufmann, San Francisco, CA, 2000.

[109] P. Sollich. Probabilistic interpretation and bayesian methods for support vector machines. In *Proceedings of ICANN'99*, pages 91–96. IEE Publications, 1999.

[110] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[111] J. A. K. Suykens and J. Vandewalle. Multiclass least squares support vector machines. In *IJCNN'99 International Joint Conference on Neural Networks*, Washington, DC, 1999.

[112] D. Tax and R. Duin. Data domain description by support vector. In *ESANN99*, pages 251–256, 1999.

[113] M. E. Tipping. The relevance vector machine. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Proceedings of Advances in Neural Information Processing Systems 12 (NIPS'99)*. MIT Press, 1999.

[114] M. E. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

[115] V. Tresp and A. Schwaighofer. Scalable kernel systems. In G. Dorffner, H. Bischof, , and K. Hornik, editors, *Artificial Neural Networks - Proceedings of ICANN 2001*, pages 285–291. Springer Verlag, 2001.

[116] V. R. University. Probabilistic discriminative kernel classifiers for multi-class problems.

[117] R. Vanderbei. Loqo: An interior point cpde for quadratic programming. Technical report, Priceton University, US, 1994.

[118] S. Vandewalle. Recurrent least squares support vector machines, 2000.

[119] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer, N.Y., 1995.

[120] V. Vapnik and O. Chapelle. Bounds on error expectation for SVM. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 261–280, Cambridge, MA, 2000. MIT Press.

[121] G. Wahba, Y. Lin, and H. Zhang. Gacv for support vector machines. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 297–311, Cambridge, MA, 2000. MIT Press.

[122] C. Williams. Prediction with gaussian processes: From linear regression to linear prediction and beyond.

[123] C. Williams. Support vector machines (course notes),university of edinburgh, uk, 2000.

[124] C. K. I. Williams and M. Seeger. Using the nystrom method to speed up kernel machines gzipped postscript. *To appear in: Advances in Neural Information Processing Systems*, 13, 2001.

[125] J. Xu, X. Zhang, and Y. Li. Kernel mse algorithm: a unified framework for kfd, ls-svm and krr. In *Proc. of IJCNN 2001*, pages 1486–1491, 2001.