Support Vector Machines for Phoneme Classification

Jesper Salomon

Master of Science
School of Artificial Intelligence
Division of Informatics
University of Edinburgh
2001

Abstract

In this thesis, Support Vector Machines (SVMs) 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. Since this task involves multiple classes, one of the main hurdles SVMs must overcome is to extend the inherently binary SVMs to the multi-class case. To do this, several methods are proposed, and their generalisation abilities are measured. It is found that even though some generalisation is lost in the transition, this can still lead to effective classifiers. In addition, a refinement to the SVMs is made to derive estimated posterior probabilities from classifications. Since almost all speech recognition systems are based on statistical models, this is necessary if SVMs are to be used in a full speech recognition system. The best accuracy found was 71.4%, which is competitive with the best results found in literature.

Acknowledgements

I would like to thank Simon King for his guidance and support throughout this project.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Jesper Salomon*)

Table of Contents

1 Introduction **1**

1.1 Summary of literature 5

1.1.1 Recurrent Neural Networks 5

1.1.2 Support Vector Machines 6

1.1.3 Overview of thesis 7

2 Support Vector Machines **9**

2.1 Introduction 9

2.2 Statistical Learning Theory 9

2.2.1 Binary Classification Problem 10

2.2.2 Empirical Risk Minimisation 10

2.2.3 Structural Risk Minimisation 11

2.2.4 VC dimension 11

2.3 Linear Classifiers 12

2.3.1 The Separable Case 12

2.3.2 Non-separable case 18

2.4 Non-linear Classifiers 20

2.4.1 The “Kernel Trick” 20

2.4.2 Training 22

2.4.3 Classification 22

2.4.4 Kernels 23

2.5 Choosing the kernel and C 24

2.6 Bias in SVMs 25

2.7 Practical Issues . . . 25
2.8 Multi-class SVMs . . . 26
2.8.1 One vs. One classifier . . . 27
2.8.2 One vs. Rest . . . 28
2.8.3 DAGSVM . . . 29
2.8.4 Binary Tree . . . 29
2.9 Conclusion . . . 32

3 Experiments **33**
3.0.1 Toolkits used . . . 34
3.0.2 The TIMIT speech database . . . 34
3.1 Binary SVM experiments . . . 35
3.1.1 Experimental setup . . . 36
3.1.2 Choice of kernel and C . . . 36
3.1.3 Including different amounts of training data . . . 37
3.2 Selecting multi-class methods . . . 39
3.3 One vs. One Classifier . . . 40
3.3.1 Introduction . . . 40
3.3.2 Practical issues . . . 40
3.3.3 First experiment: Effects of kernels . . . 41
3.3.4 Adding more features . . . 42
3.3.5 Adding context frames . . . 43
3.3.6 Consistency experiment . . . 45
3.3.7 Last experiment: Including additional training data . . . 46
3.3.8 Generalisation performance of the One vs. One classifier . . . 48
3.3.9 Conclusion . . . 49
3.4 DAGSVM classifier . . . 50
3.4.1 Results and conclusions . . . 51
3.5 Binary tree classifier . . . 51
3.5.1 Introduction . . . 51
3.5.2 Implementation . . . 52
3.6 Conclusion . . . 54

4 Probability outputs **56**
4.1 Estimating posterior probabilities 56
4.2 Conclusion 58

5 Conclusion and future work **60**

A Kernel Requirements **63**
A.1 Symmetry Condition 63
A.2 Mercer's Theorem 63

B Phonemes in the TIMIT speech database **65**

C Binary Tree Structure **67**

Bibliography **69**

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).

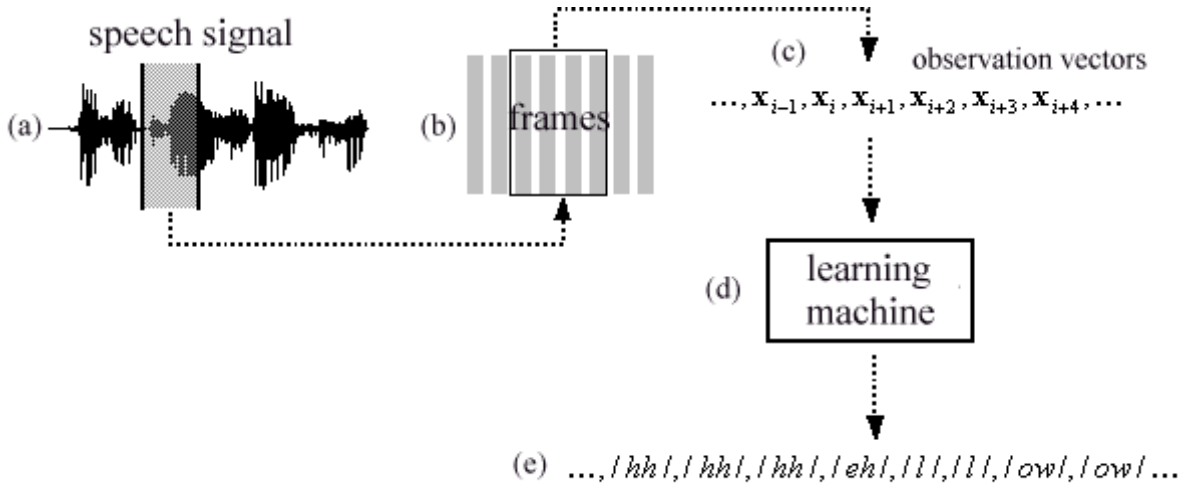

Figure 1.1: Overview of frame classification.

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 machine for the task is Support Vector Machines (SVMs). SVMs are a fairly new technique for pattern recognition, created by Vapnik ([60]). They have been applied to many different problems, and have been very successful in areas such as face recognition ([45]), text-categorisation ([28] and [13]), time-series prediction ([42]), and hand-written digit recognition ([55]). 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 ([56]) and Novelty detection ([58]). 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 (see figure 1.2), where the data can be separated using a linear hyperplane.

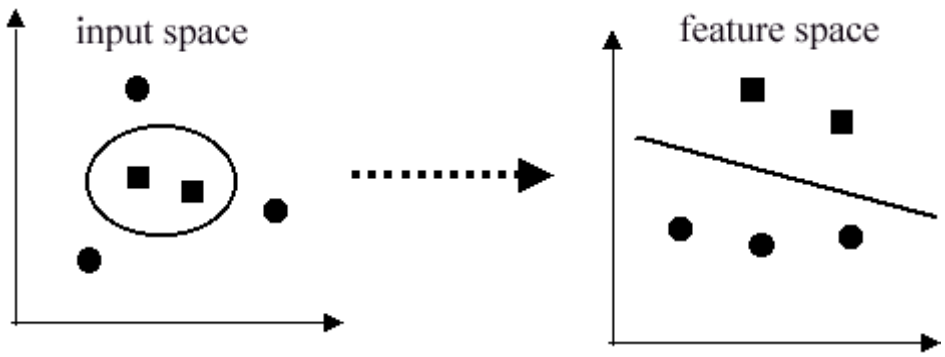

Figure 1.2: The mapping of non-linearly separable training samples from input space to a linearly separable representation in feature space.

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 Min-*

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

imisation principle, which seeks to minimise the upper bound on the generalisation error ([60]). 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 the SVM to evaluate scalar inner products in feature-space. This can be very complicated and computationally expensive if the feature-space is of much higher dimensions 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.

By using SVMs in speech recognition, it is hoped that some of the performance gains shown by SVMs on other tasks might be mapped over to this domain. Limited work has been done to apply SVMs to the problem. The results so far have been promising, but several significant hurdles muct be overcome if SVMs are to be viewed as a viable and competitive alternative to the established methods. Problems include:

- *The problem of multi-class classification.* 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 an SVM 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 SVM. 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 the SVMs 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 SVMs 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 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[2].

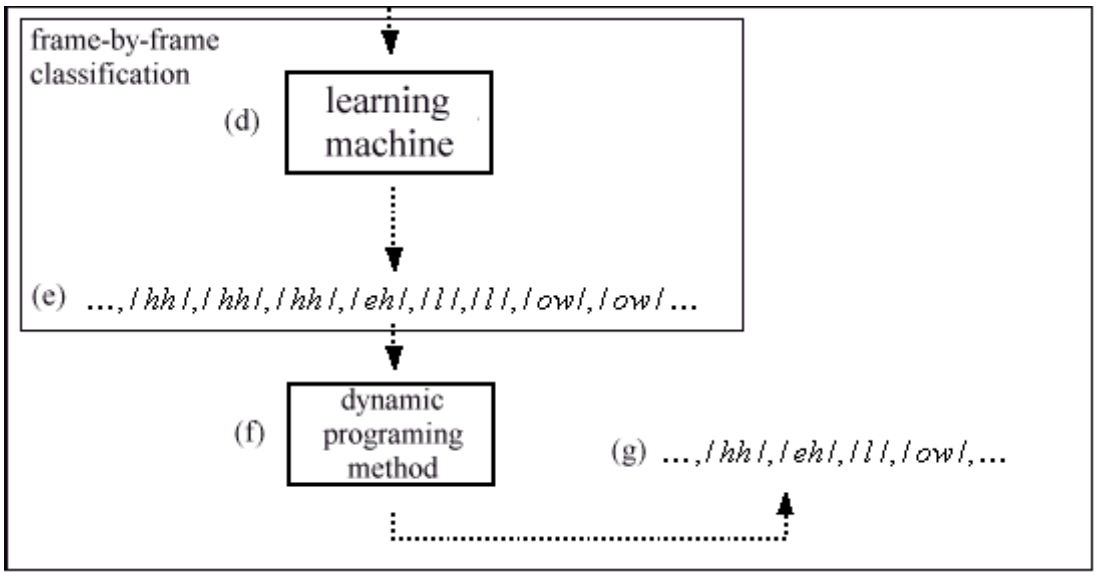

Figure 1.3: Difference between phoneme recognition and frame-by-frame classification. The small box encloses the task of a frame-by-frame classifier, and the big box encloses the task of a phoneme recogniser. After the frames have been classified in (e), the dynamic programming method (f) finds the most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

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

individual phonemes a sentence is composed of. This differs slightly from *frame-by-frame* classification in the sense that phonemes typically have durations much longer durations[3]. 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 SVMs.

1.1 Summary of 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 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 ([50]) created a phoneme recognition system that has produced the best phoneme recognition results reported in literature[4]. 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%[5]. Until recently, this was the best-reported result on the task. The system utilized a time-delayed recurrent neural network (RNN), a variant of

[3]Typically, *frames* are extracted with 10 ms intervals. In average, phonemes lasts 65ms ([25]).

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

[5]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).

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 ([5]). 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 Support Vector Machines

A few efforts have been made to apply SVMs to the task of speech recognition.

In [9], Clarkson created a multi-class SVM system to classify phonemes[6]. 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 was retained, and thus the problem was considerably easier than that of frame-by-frame classification[7]. 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 [19], 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 [9]. The performance of the system was compared with a standard HMM system where Gaussian Mixture Mod-

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

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

els (GMMs) were used instead. On the OGI Alphadigits speech database ([10]), 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 [6], 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 hand-picked subset of the TIMIT speech corpus ([20]).

In addition to recognition, SVMs have also been used in other speech and sound applications. These include speaker identification ([52]), musical instrument classification ([38]), utterance verification ([36]), and web audio classification ([41]). 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 in speech recognition. Some have discussed and attempted to tackle some of the problems involved, but the SVMs 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 SVMs 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 organised as follows:

In chapter 2, I explain the concepts behind linear and non-linear SVMs, feature space, and the role of the kernel. Additionally, I describe several techniques for extending the inherently binary SVMs to multi-class problems.

In chapter 3, experiments using three 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.

Finally, chapter 5 will summarise findings and give directions for possible future research.

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 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 describe the most common methods for extending the binary SVMs to handle multiple classes.

2.2 Statistical Learning Theory

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 loosely based on [4], [63], [6], and [22].

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 $f(\mathbf{x}_i) = y_i, \forall \langle \mathbf{x}_i, y_i \rangle \in D$.

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

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

where N is the size of the training set and α 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. This is because the minimisation principle does not consider the capacity of the machine.

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 α to learn the mapping $\mathbf{x} \mapsto y$. This is done by minimising the expected risk:

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

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. [61] showed that the upper bound on this error is:

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

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 $\mathbf{x} \mapsto f(\mathbf{x}, \alpha)$ 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[2]. 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]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.

2.1. The 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 [60].

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

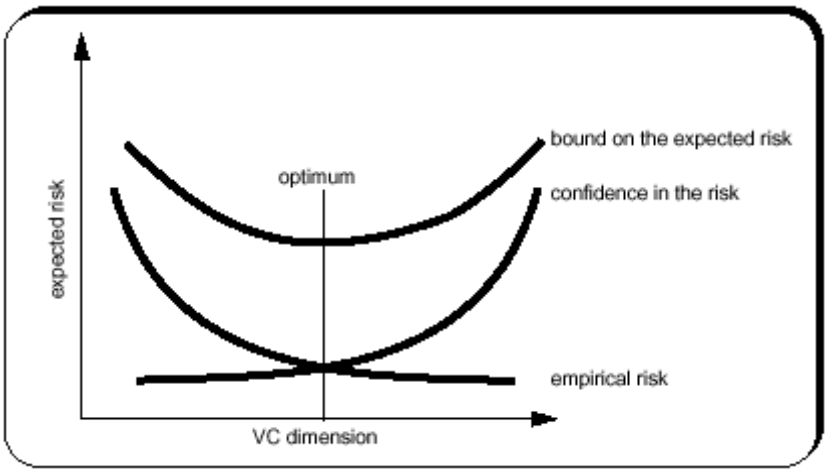

Figure 2.1: Relation between empirical risk, VC dimension and expected risk (from [18]).

2.3 Linear Classifiers

In the following two sections, SVMs will be formulated as a maximum margin classifier[3]. There are two cases of linear classifiers to consider. In section 2.3.1, the case where a perfect mapping $\mathbf{x} \mapsto f(\mathbf{x}, \alpha)$ can be learned will be explained. Subsequently, in section 2.3.2, the case where a perfect mapping is unattainable is described.

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_- .

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

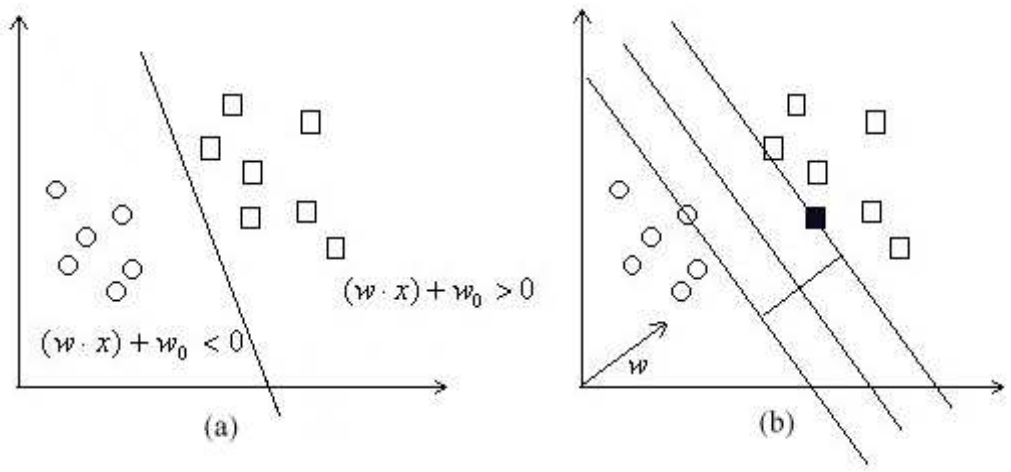

Figure 2.2: (a) a separating hyperplane. (b) The hyperplane that maximises the margin of separability

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

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

where w is a weight vector and b the offset from origin.

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.

2.3.1.1 A Maximum Margin Separating Hyperplane

There are numerous values of $\{\mathbf{w}, b\}$ that 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 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:

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

Looking at figure 2.2(b), if we define the margin as $d_+ + d_-$ it is obvious that for the margin to be maximal, $d_+ = d_-$. 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 $x_+ * w + b = 1$ 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 can be reformulated as:

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

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 ($d_+ \neq d_-$).

2.3.1.2 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

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.

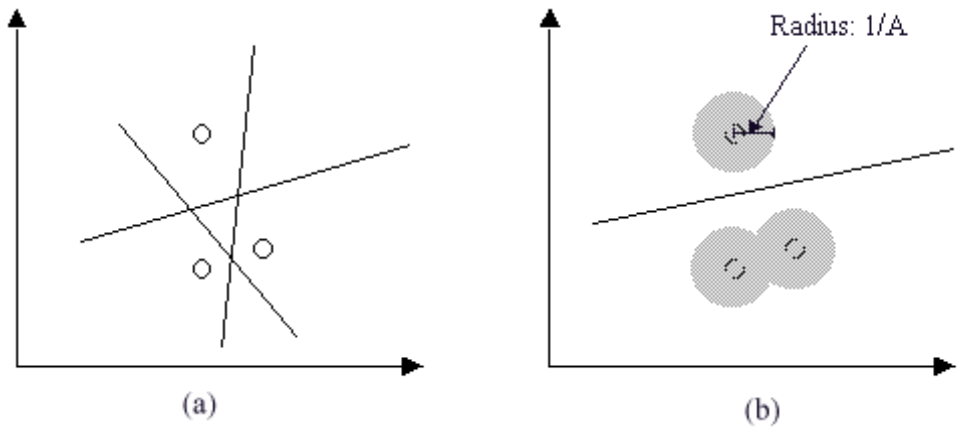

Figure 2.3: (a) All hyperplanes that separates the data points are shown. (b) The number of possible separating hyperplanes is reduced when an upper bound A is put on $\|\mathbf{w}\|$.

From [60], 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.

Therefore, 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.

2.3.1.3 Lagrangian formulation

The task to solve is a minimisation problem of $\|\mathbf{w}\|^2$with a set of inequality constraints from from equation (2.7)[4]. The theory of Langrangian multipliers is well known to efficiently solve this problem ([3]).

The Lagrangian formulation of the minimisation problem is:

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

Subject to 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 inequality constraints into the objective function. The resulting dual formulation is:

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

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

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 $\alpha_i \geq 0$, 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.

2.3.1.4 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 **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, $\mathbf{x} \mapsto f(\mathbf{x}, \alpha)$, 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 $\|w\|^2$. Misclassifications are considered by adding a "slack" variable $\xi \geq 0$ for each training example, and require that:

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

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[5]. On figure 2.4, an overview of a non-separable case is shown.

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

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

where C is a regularisation parameter used to control the relation between the slack variables and $\|w\|^2$. 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

[5]Other error measures exist, e.g. the squares distance.

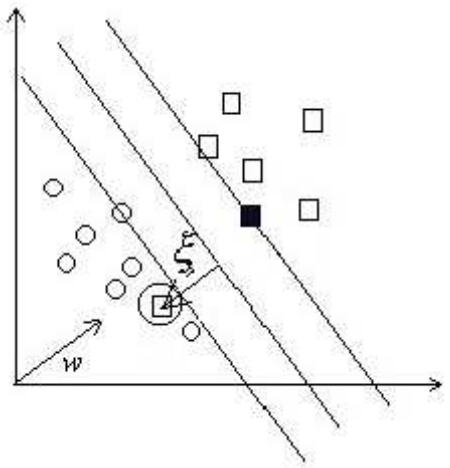

Figure 2.4: A non-separable case. The encircled data point is misclassified and thus has a positive ξ.

disappear from the dual Langragian problem ([4]. This objective function of the dual formulation becomes:

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

Subject to constraints:

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

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

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 [23]. 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 a non-linear mapping Φ is used, the resulting hyperplane in input-space will be non-linear.

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

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

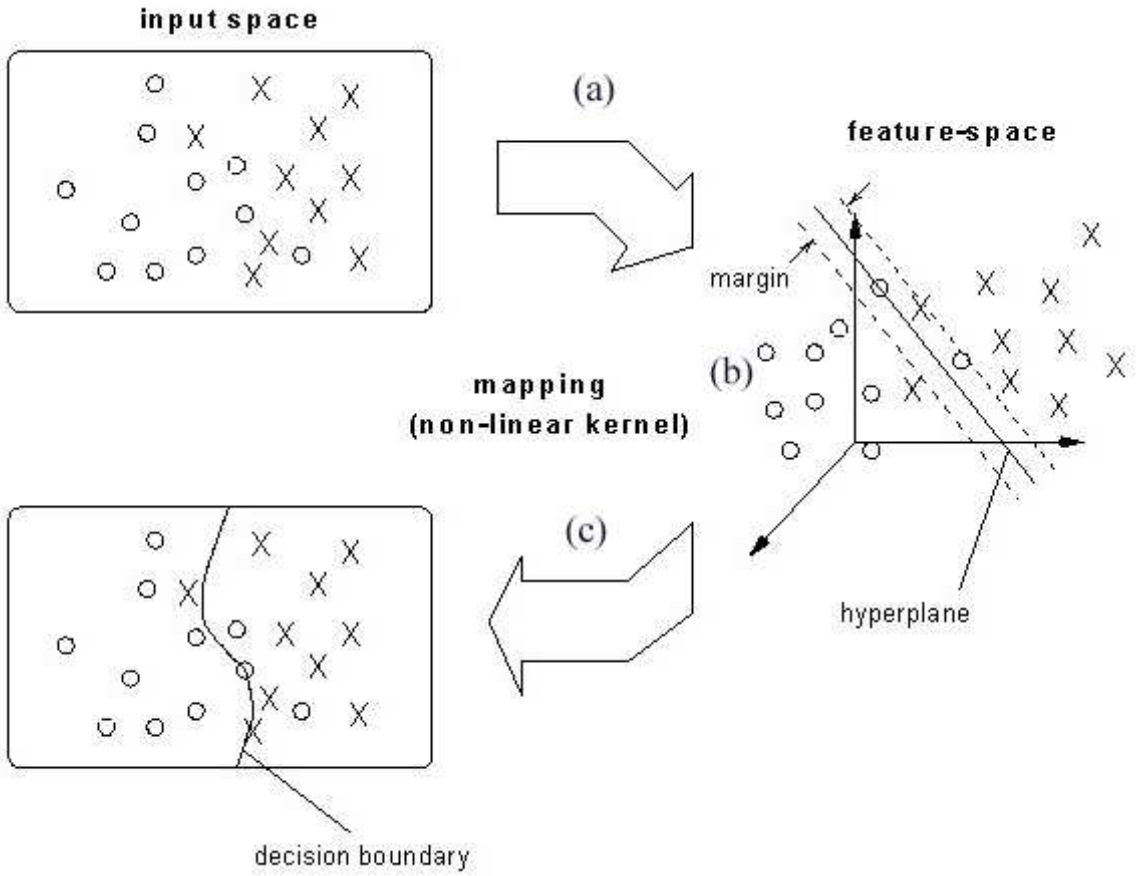

Figure 2.5: The role of the kernel (from [43]). (a) The data is mapped from input-space to feature-space by a mapping Φ. (b) The optimal separating hyperplane is found. (c) The hyperplane is mapped back down to input-space, where it results in a non-linear decision boundary.

Guyon ([23]) showed that an old trick by [35] 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 in equation (2.23) that 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:

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

The functional representation is called a *kernel*, and SVMs are a member of the broader class of kernel methods [54]. If the feature-space is of much higher dimension than the input space, this implicit calculation of the dot-product 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[6].

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:

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

Subject to same constraints as in the linear case:

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

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

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:

[6] 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.

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

where b the offset of the decision boundary from origin.

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

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

2.4.4 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 ([3]).

these conditions are described in details in Appendix A.

Some popular kernels includes:

- The homogenous polynomial kernel:

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

  If input-space is $\Re^N$, this kernel maps the samples into a fixed $(N+d)$-dimensional space[7]. The resulting the decision boundary will be polynomial ([4]).

- The Gaussian kernel:

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

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

This kernel centres a Gaussian with variance σ^2[8] 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 ([22]).

- The Linear kernel:

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

which is results in exactly the same objective function as in equation (2.23). Note, that this kernel is simply a specific case of the Polynomial kernel with $d = 1$.

2.5 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 ([22]). However, even though the VC dimension is describes the complexity and flexibility of the 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 the kernel selection.

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 on a validation set.

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

2.6 Bias in SVMs

Recall the training process of the linearly separable case (section 2.3.1.1). 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 is learned in the training process.

However, in some cases it is desired to have a decision boundary that is *not* placed learned in the training process. 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:

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

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 ([29]).

2.7 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[9], 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* ([44]). This technique spliits up the training set into smaller sub sets, 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[10] are kept, and the remaining training examples are

[9] For example LOQO ([59]).

[10] Training examples with $a_i > 0$ in equation (2.30).

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* ([46]). In this optimisation procedure, the training problem is decomposed into tiny tasks of optimising only two of the a_i s in equation (2.30). The remaining a_i s are kept fixed, and these two values are easy and fast to find[11].

These two techniques are implemented in the most popular toolkits[12]. However, even though they provide drastically reduced training times, the relation between amount of training data and training time still remains non-linear. 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 ([29]).

2.8 Multi-class SVMs

Many real-world data sets involve multiple classes. Since SVMs are inherently binary classifiers, techniques are needed to extend 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 [16], [39], [53], [21], [26], and [15]. This section will not try to describe all of these, but concentrate on the ones that have shown to produce good generalisation performance in practice.

[11] To go into more details of this optimisation procedure would require an extensive description. Refer to [46] for a detailed description.

[12] SVM-Light and SVMTorch. See [29] and [12].

2.8.1 One vs. One classifier

The One vs. One classifier is a system proposed by Friedmann ([16]), and has become the most popular and successful multi-class SVM method. The principle behind the method is very simple. It creates a binary SVM for each combination of classes[13] possible, and each unseen example are classified to the class that "wins" most binary classifications[14]. This method is also called the *voting scheme*, since each binary SVM classification assigns one "credit", or "vote", to one of the two competing classes.

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 used 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 SVM model of $\{c_i, c_j\}$ as:

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

An unseen example, **x**, is then classified as:

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

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 subproblems is that the different decision boundaries can be creates for each class-pair. This potentially results in a very complex decision boundary. Furthermore,

[13] A class denotes a target label. "Classes" denotes the set of possible target labels. These terms will be used inter-changeably throughout the thesis.

[14] 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)

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 there are $K-1$ binary models per class.

- 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. Correspondingly, the number of binary SVMs also makes classification slow as all need to be evaluated before a decision is made.

Nevertheless, this classifier has produced some of the best results on multi-class tasks found in SVM literature ([39]).

2.8.2 One vs. Rest

In this classification scheme, K binary SVMs are built. Each attempts to build a decision boundary separating one class, c_i, from the rest. Creating the models are accomplished by assigning the label "+1" to c_i, and the same binary target label, "-1", to all remaining classes.

Classification of an unseen example, $\mathbf{x}$, is done by computing the function value of equation (2.33) for each binary SVM model. This differs from the previous classification scheme, which use equation (2.34). The motivation for using this decision function is to avoid "draws". I.e. being unable to predict a winner if more than one c_i "wins" it's corresponding binary SVM. Instead, the class that maximises the value of equation (2.33) will be chosen. The resulting classifier is:

$$f_{1vsR}(\mathbf{x}) = \arg\max_i f_i(\mathbf{x}) i = 1, \ldots, K \tag{2.41}$$

where $f_i(\mathbf{x})$ is the SVM model separating the i th class from the rest.

- The advantage of this method is the few SVMs involved. It only needs to build K models, and consequently evaluation is much faster than the One vs. One classifier.

- However, it has several drawbacks. By only building K classifiers the resulting decision boundary can never be as complex as the boundary of the One vs. One

classifier. Furthermore, since all classes are involved in each SVM, training the SVMs can be very time consuming. Finally, in each binary SVM it is often difficul to isolate one class from the rest. All training examples are weighted equally, and the uneven distribution of training examples between the isolated class and the remaining classes makes separation difficult. To improve separation an explicit bias towards c_i must be used (as described in section 2.6).

2.8.3 DAGSVM

DAGSVM stands for *Directed Acyclic Graph SVM*. It is a multi-class method proposed by Platt ([27]), 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 i th and j th 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.

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

2.8.4 Binary Tree

The last multi-class method to describe is the binary tree classifier. It is based on a hierarchical structure, and can best described as a SVM version of a Decision Tree ([48]). The idea of growing a tree structure from data is very common in machine learning, and used in learning machines such as Decision Trees, k-Nearest Neighbour

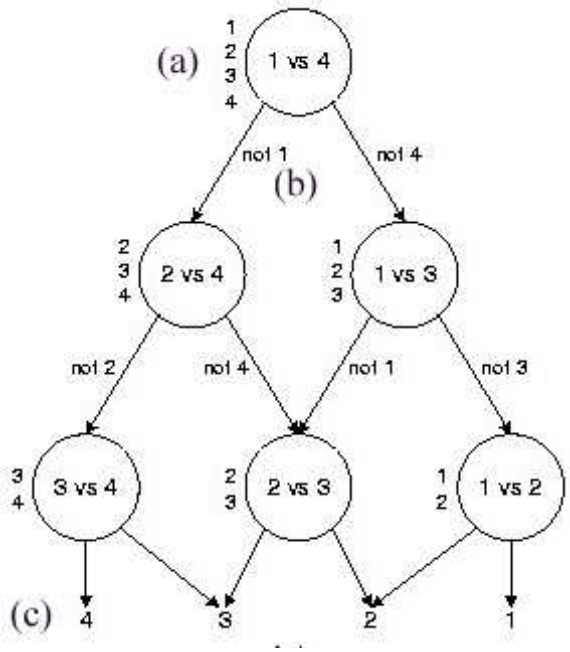

Figure 2.6: Decision graph for a classification task with 4 target classes. Starting at the root node (see (a) on the figure), the SVM belonging to the node is evaluated, using equation (2.12). (b) It moves down the tree to either left or right child depending on the outcome. (c) It repeats this process until a leaf k is reached, and assigns the unseen example to the k th class (from [27]).

Trees ([40]), and CART models ([30]). However, although this classifier seems apparent considering the binary nature of SVMs, I have not found this classifier mentioned in SVM literature.

The classifier I propose is a rooted asymmetrical tree structure as shown on figure 2.7. In contrast to the DAGSVM classifier, the nodes in the binary tree can contain multiple classes, splitting these up into two equally sized subsets (see (a) on figure). Each child node contain another binary SVM with classes form one of the two subsets (see (b) on the figure). As we move down the tree, the number of classes contained in each node decreases until only one class remains (see (c)).

The problem of growing trees is discussed extensively in literature (for example [33]). Several algorithms exist for this task, for example K-means ([40]), and Additive

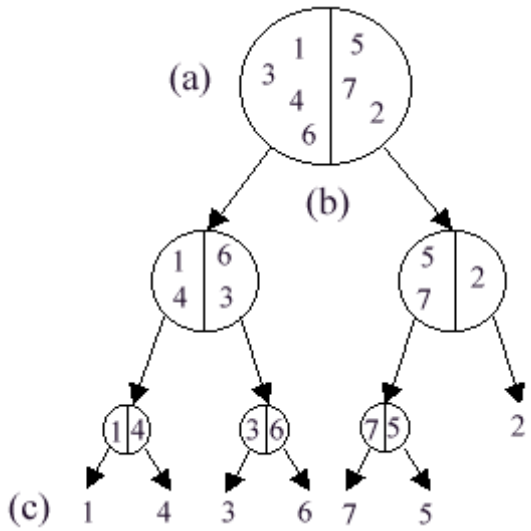

Figure 2.7: Schematic of a binary tree on a 7-class problem.

Similarity Trees ([51])[15].

To classify an unseen example **x**, consider figure 2.7 once again. Classification is completed as follows. (a) First, the root node the binary SVM is evaluated. (b) Depending on which side of the decision boundary **x** falls, it moves either down to the left child or the right child. (c) This process is repeated until a leaf is reached. This leaf indicates the classification of **x**.

- It is hoped that the generalisation ability shown by decision trees can be translated to the case of multi-class SVMs. Even though speed is not a concern in this thesis, the binary tree classifier has an extremely fast classification phase. If the tree structure is symmetric, it requires only $2 * lnK$ evaluations to classify unseen examples. This is substantially less than the evaluations needed by the above methods.

- However, this can also be viewed with pessimistic eyes: The separating hyperplanes in the top nodes boundary cannot be very detailed as they need to include all classes. Additionally, as in the DAGSVM classifier, if *one* binary misclassification happens during classification, the unseen example will be misclassified.

[15]Note that this structure does not need to be symmetrical. Depending on the inter-relation amongst the classes, one can opt to create an asymmetrical and deep tree.

This may make it less stable than the One vs. One classifier.

2.9 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 on 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[16]. Furthermore, a more detailed theoretical analysis of the generalisation bounds of binary SVMs could lead to a better understanding of their generalisation performance (see [1]).

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

[16] Rrefer to [24] 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.

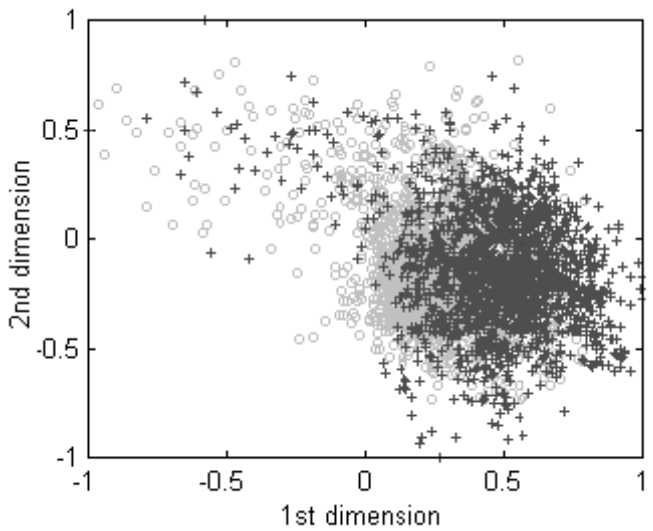

Figure 3.1: Plot of the first two dimensions from /aa/ and /ae/ vowel speech frames.

In the following sections four groups of experiments will be described. First, anal-

ysis of models trained on class-pairs is performed to learn how to get the best generalisation performance out of each individual SVM. Subsequently, three groups of multi-class experiments will be carried out. These are related in the sense that they each utilize one of the multi-class classification methods described in section 2.8.

3.0.1 Toolkits used

Throughout the experiments, I used the toolkit SVM-*Light* ([29]). It implements the *Chunking* algorithm described in section 2.7. 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 ([11]). This is faster due to the implementation of the *Sequential Minimal Optimisation* algorithm also described in section 2.7.

3.0.2 The TIMIT speech database

The data set used for the task is the TIMIT database[1] ([20]). 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 ([49]).

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

[1] The description of the TIMIT database is based on [32].

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

chosen[3].

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

3.1 Binary SVM 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 parameters of 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 2.4.4. 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 2.7, 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

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

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 fature 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 2.4.4. 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.

[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.

Nevertheless, techniques for automatically learning the SVM parameters were investigated. Several such schemes exist in literature ([31], [34], and [62]). I examined some of these, but unfortunately, the increase in performance by employing these methods were marginal (mostly less than 0.1%). Furthermore, the run-time needed far exceeded the run-time of the exhaustive search.

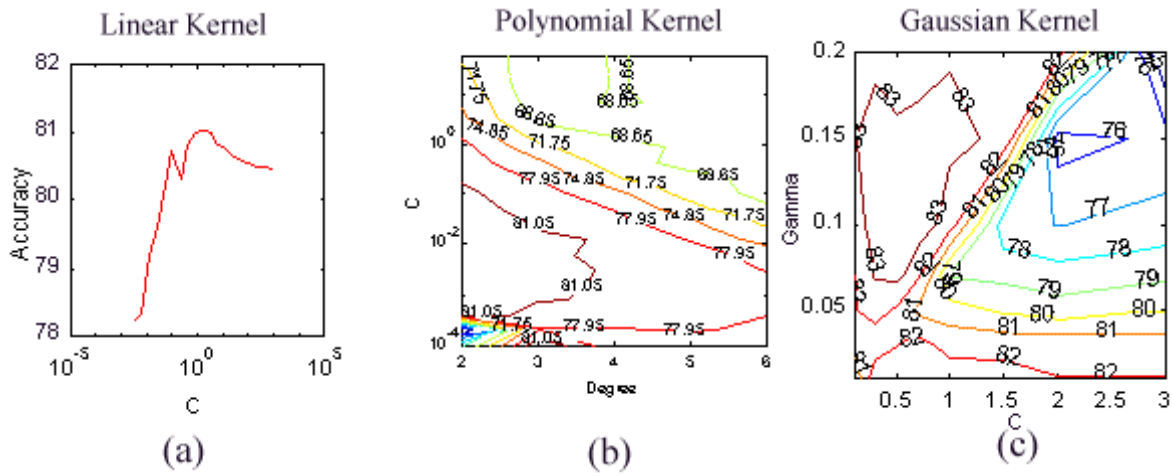

Figure 3.2: Effect of C and kernel parameter on validation accuracy. Plot (a) show the resulting performance of the linear kernel when varying C. Plot (b) displays performance of the polynomial kernel when varying C and degree d . Finally, plot (c) shows performance of the Gaussian kernel when varying 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.

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 tine. Since the training time increases non-linearly 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.

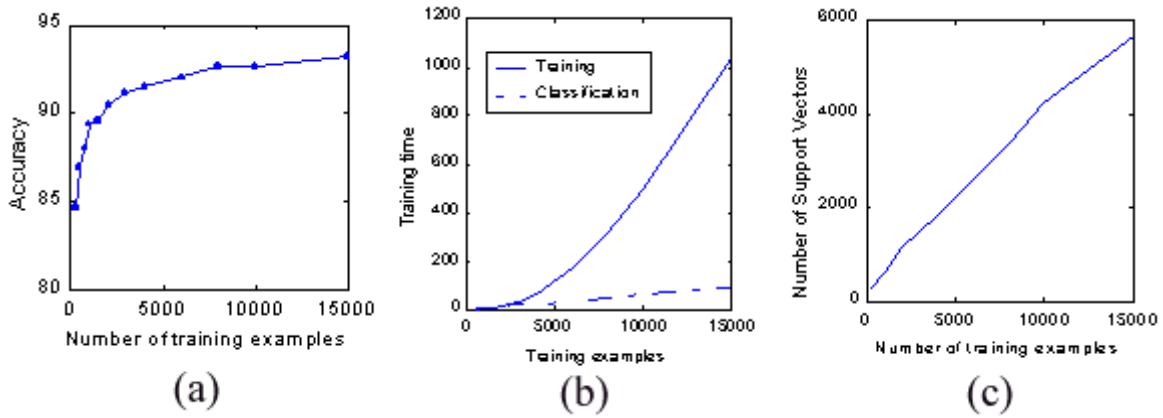

Figure 3.3: The effect of varying the training set size. Plot (a) shows the increase in generalisation performance when varying the size of the training set. As expected, the performance increases when the amount of training data increases. Plot (b) show that the relation between training set size and 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.

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:

- The Gaussian kernel seems to offer the best generalisation performance. This was expected: the linear and polynomial uses a feature space with a fixed number of dimensions. In contrast, the Gaussian kernel has the potential to map the data into 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 kernels 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.

- 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 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).

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 2.8, I described four different methods for extending the binary SVM to handle multi-class data. Due to the computational costs of running the multi-class experiments, I have chosen not to use the One vs. Rest classifier (described in section 2.8.2). Although this is one of the most commonly used methods, the performance on speech data seems to be lacking. In preliminary experiments, this method produced substantially worse results than the remaining classifiers, and is found to be incapable of properly separating the data. This conclusion is supported by results found in [9] and [6].

This leaves the One vs. One, DAGSVM, and the Binary Tree classifiers. In the following sections, I will start out describing the One vs. One classification experiments in details. The remaining classification schemes will be explained much more briefly, since they 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

Recall the description of the One vs. One classifier in section 2.8.1: this classifier creates one binary SVM for each combination of classes, resulting in $K(K-1)/2$ SVMs. When an unseen example is classified, all SVMs are evaluated. For each SVM, 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 ([4], [3]). In numerous cases, including frame classification, this classifier has shown superior generalisation performance ([9], [6]).

The task in focus is frame classification. It has 40 possible target labels, where each target corresponds to a phoneme class. This results in $40 * (40-1)/2 = 780$ binary SVMs. This is a considerable number of SVMs. 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[9]. This should still be enough to give a reasonably accurate measure of performance. 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.

[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] 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.

3.3.3 First experiment: 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.

3.3.3.1 Experimental setup

To begin with, I chose to include only 1000 training examples per class, resulting in 2000 training examples per SVM. 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 had the 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.

3.3.3.2 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 [50] and 74.8% in [5]). The explanation can perhaps be found in the binary

[10]Note on notation used in table. MBA stands for *Mean Binary Accuracy*. It is the average of all accuracies found from evaluating a 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.
MA1 and MA2 denote *Multi-class Accuracies*. MA1 is found by evaluating the classifier on 4000 test examples per class. MA2 is found by weighting the results from MA1 with the phoneme frequencies in the full TIMIT test set. Intuitively, MA2 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.

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 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 [43], the 13 coefficients used has a large in-class variance. Thus the risk of large differences between validation and test sets is high, and leads to poor performance.

Kernel	**MBA Val**	**MBA Test**	**MA1**	**MA2**	**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 (MA1 and MA2), 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%.

3.3.4 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. The can be computed over several frames to comprise the dynamics of the speech and help discriminating between fast and slow changing phonemes. The features are commonly used in speech recognition, and are typically found to improve classification results ([49]).

3.3.4.1 Experimental setup

When adding the new features, the feature vectors contained the 13 existing MFCC coefficients, 13 first order derivatives and 13 second order derivatives. 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	MA1	MA2	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.

3.3.4.2 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. I.e. it demonstrates the theoretical claim that SVMs counteracts the *curse of dimensionality* (from section 2.3.1.2).

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 [18] and in the extensive examinations of [37] and [25]. Investigations into adding additional context information seem reasonable.

3.3.5 Adding context frames

How can we include more context information about each frame? The answer lies in the neighbouring frames. Speech is fairly slowly changing, and each phoneme lasts on average 65ms [25]. 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.

3.3.5.1 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, 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 **x**	$\lvert\mathbf{x}\rvert$	MBA Val	MBA Test	MA1	MA2	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.

3.3.5.2 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 [5], may seem substantially better than the 59.8%/63.7% found in this experiment. However, the result should not be dismissed so easily. [5] also report a 60.9% frame accuracy using only 10% of the training data. Since my experiments are based on approximately the same amount of training data, I believe them to be very promising.

3.3.6 Consistency experiment

So far, the best-found result has been based on a total of 160000 examples from the TIMIT test set (4000 examples per class). Since this is only a subset of the full test set, we need to check the solidity of the result, to certify that it can be reproduced on different test sets.

3.3.6.1 Experimental setup

Three new separate subsets were created from the TIMIT test set, containing 1000 examples per phoneme class[11]. This is in addition to the test set used in the previous experiments. The sets were tested on the best models found thus far[12].

Test set	MBA Val	MBA Test	Overall accuracy
Base	95.6	95.0	59.8/63.7
1	95.2	94.9	59.4/63.1
2	95.8	95.2	60.1/64.2
3	95.6	95.1	59.8/63.8
Overall	95.5	95.1	59.9/63.8

Table 3.6: Results from consistency experiment.

[11] As mentioned before, some classes do not have sufficient examples in the full set (Refer to appendix B for an overview of the TIMIT speech corpus). In these case, the same examples were used in all sub sets.

[12] Using 39 features per frame, two context frames on each side of the target frame, and the Gaussian kernel.

3.3.6.2 Results and conclusions

The results from the tests are shown in table 3.6. They show that the variances in both binary and overall accuracies are marginal, proving that the previous found results are reliable.

3.3.7 Last experiment: Including additional training data

In the previous experiments, each binary SVM has included only 1000 training examples per phoneme class, resulting in 2000 training examples per SVM. 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 SVMs could improve if additional training data is added. Consequently, this could lead to better generalisation performance.

3.3.7.1 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. 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.

3.3.7.2 Results and conclusions

The results from the two experiments are shown in table 3.7, 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%

Examples per class	MBA Val	MBA Test	MA1/MA2	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	-	**96.1**	55.9/**70.8**	**86.3/93.5**
16000 (full)	-	**96.1**	70.6	86.1/93.4

Table 3.7: 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.

accuracy found on the full test set is better than the second best result found in literature (70.4% by [50]). This proves that SVMs is a viable and capable technique for frame classification.

Another important observation can be made. Notice how the unadjusted overall performance (MA1) *decreases* when the amount of training data increases. Although this may come as a surprise, a logical explanation exists:

As mentioned in section 3.0.2, some phonemes are very infrequent and have few training examples 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 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. Hence the improved classification performance on the full test set. However, as a result of this bias, the smaller classes will "lose"[13] more classifications, and the unadjusted classification rate will decrease.

[13] As opposed to "winning".

3.3.8 Generalisation performance of the One vs. One classifier

To measure the generalisation performance of the One vs. One classifier, consider how the overall accuracy is related to the binary accuracy of the 780 binary SVMs. It seems logical that if *all* 780 binary SVMs have a 100% test accuracy, this will 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 3.4.

The plot can be perceived in two ways. On the hand, the figure shows a rather depressing fact: Even large improvements in binary accuracies does not lead to a great improvement in overall accuracy. On the other hand, looking at the estimated trend line, even a slight increase in binary accuracy should lead to a substantial increase in overall accuracy.

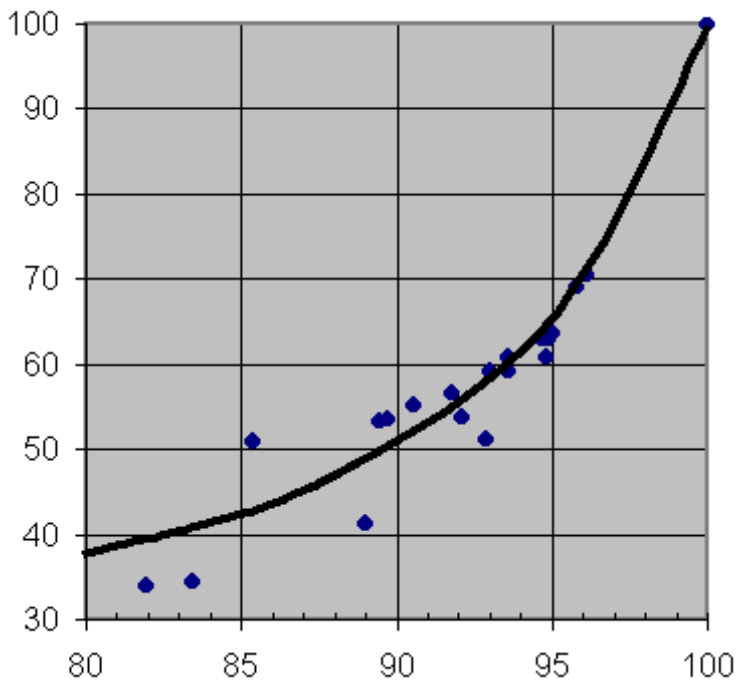

Figure 3.4: Binary mean accuracy (x axis) vs. overall accuracy (y axis) using the voting scheme.

3.3.9 Conclusion

In the above experiments we have seen that the One vs. One classifier offers great 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 70.6% frame classification rate is competitive with the second best result reported in literature (70.4% in [50]).

Conclusions can not only be made about the One vs. One classifier, but about the SVM method in general. Average binary accuracies 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 SVMs required to build the multi-class model. The resulting model has just under 2 million components[14], 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.

A new multi-class classifier which builds on this principle is examined in the next section.

[14] Total number of support vectors multiplied by number of features in the feature vectors.

3.4 DAGSVM classifier

3.4.0.1 Introduction

Platt's DAGSVM algorithm [27] was described in section 2.8.3. To briefly recapitulate, 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 is done, starting at the root node, by moving down the tree following the left or right branch of a node depending on the outcome of each node's SVM. The leaf it reaches indicates the classification of the unseen examples.

In [26] and [27] 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.

3.4.0.2 Implementation issues

[27] 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.

[27] 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[15].

[15] In the sorted list, the phonemes were sorted by their phonological groupings. I.e. vowels/semi-vowels, nasal/flaps, stops, and fricatives (see [8] for details).

3.4.0.3 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 3.3.7). 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 3.8: 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.

3.4.1 Results and conclusions

The results can be found in table 3.8[16]. 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 ([27] and [26]). Appreciably, as a result of sorting the list the result is slightly better than the previously best result (from section 2.8.3). This reaffirms the SVM method's capability as a phoneme classifier.

3.5 Binary tree classifier

3.5.1 Introduction

The last multi-class scheme to be evaluated in the thesis is the Binary Tree classifier. It was described in section 2.8.4 and to briefly summarise, it uses a rooted tree structure with nodes containing binary SVMs. Unlike the DAGSVM algorithm the nodes in the tree can contain multiple classes. These classes are separated into two subsets and a model is trained to build a decision boundary between them. In classification, starting

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

at the root node, each node's SVM is evaluated, and depending on this classification, it moves down the left or right branch of the node. This is repeated until a leaf is reached - which indicates the classification of the unseen example. A schematic of the structure can be found on figure 2.7.

Since I have not met any literature covering this method for use in SVMs, I have not gotten any results to compare with.

3.5.2 Implementation

One important part of the binary tree classifier is the problem growing the tree structure. Unlike the DAGSVM algorithm, where changing the tree structure only had little impact on the overall accuracy, it is vital to create a structure that separates the classes optimally. Hierarchical clustering algorithms can solve this problem. They attempt to grow structures that maximise the separation between the data points.

In this paper, I have chosen to use an algorithm called AddTree ([14]). It is an implementation of the *Additive Similarity Trees* algorithm by [51]. To describe it briefly, it grows several small trees with 4 leaves each and combines these together to create the complete tree (see [51] and [14] for details).

There were one drawback of employing this algorithm: Even though it is an optimised version of the original algorithm by [51], the run-time scales $O(n^3)$ with the amount of training data. Consequently, it was not feasible to grow tree structures based on more than a tiny subset of the training data (less than 50 examples per class), and the resulting tree was deemed unsatisfactory. Hence, I used an alternative data set to grow the structure.

In the One vs. One classifier (section 2.8.1) 780 binary SVMs were built, each separating two phoneme classes. The performance on a validation set was evaluated for each of the 780 class-pair. This resulted in 780 values describing the separation ability amongst the classes. These results were used in this section to grow the tree structure seen in Appendix C[17].

[17] Looking at the tree structure, notice that the tree has *three* branches from the root. This was reduced to two branches, such that the middle branch (containing /ng/,/n/, and /m/ phoneme classes) was placed as the upmost branch in the left part of tree.

3.5.2.1 Experimental setup

A binary SVM model was trained for each of the nodes in the tree. The training data from the classes contained in a node were clustered into two groups, separating the classes according to which sub tree they belonged to.

The setup used the same user-defined parameters as in the previous experiments[18]. The issue of how much training data to use was more problematic. This had to be restricted depending on the number of classes to separate in each node. E.g. for the root node, where all 40 classes were included, the number of training examples per class was restricted to 1000. Still, this lead to 40000 training examples in total. The remaining nodes (with fewer classes) could include more training data per class. Overall, for each SVM, I kept the number of training examples constant to 40000, and adjusted the contribution from each class accordingly. Given that a SVM with 40000 training examples take a long time to train, I opted not to search through parameter space to find good parameter sets. Instead, I based the choice of parameters on the "best" parameters found in the previous experiments.

Testing was performed on a subset containing 4000 examples per class.

MBA Test	MA1	MA2
91.3	56.6	65.5

Table 3.9: Result from the binary tree experiment. The first column is the average binary accuracy for all nodes, and the second is the actual accuracy found on the test set. The last column is the estimate for the accuracy on the full test set.

3.5.2.2 Results and conclusions

The results from the experiment is found in table 3.9. Even though the classifier only uses 17% of the training data used in the 16000-examples One vs One and DAGSVM experiments, the result is less than 6% worse than the previous best reported result. This is impressive considering it only needs 3-10 evaluations to classify a frame.

Looking at the tree structure in Appendix C, it is very interesting to note that the structure closely resembles phonologically based feature systems found in literature.

[18] Using the Gaussian kernel and two context frames on each side of the target frame.

For examples, the *Sound Patterns of English* found in [7] uses 13 binary features to describe each phone. These features are based on the way the phoneme sounds are produced, such as *nasal* (for using the nasal cavity), *continuant* (for continuing sounds as opposed to fricatives), *round* (describing the rounding of the lips), etc. Looking at the upper parts of the tree structure, the phonemes are grouped by the binary features *nasals* (middle branch), *voiced* (right branch), and *consonantal* (left branch). Moreover, further down the tree the phonological based separations are still noticeable. E.g. it splits the branch containing *consonantal* phonemes into *voiced* and *unvoiced* phonemes.

These similarities between the data-grown tree structure and phonologically based groupings demonstrate two points: Firstly, the structure seems reasonable since phonemes which have similar "voicing" are found in same parts of the tree. Secondly, the groupings found in literature translate well into groupings derived from the data-driven method.

3.6 Conclusion

The above experiments have proven that SVMs are a capable and well-performing learning machine for frame classification. 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 [5], it is slightly better than the 70.4% reported in [50]. This is noteworthy given that the latter has been in development for years.

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

The amount of binary SVMs needed in training made the experimental process very extensive. Combined with the long time required to train each SVM, the run-time of each experiment seriously affected the progress rate[19].

[19]Many experiments had to be delayed because their parameter choices were to be based on experi-

To solve the restrictions on training set size, some variants of the SVM algorithm found in literature was examined ([17] and [57]). These were designed to alleviate the problem, and claimed to function with extreme amounts of training data. However, these were found to return inferior results[20].

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 2.6 that it is possible to explicitly incorporate a bias into the SVMs. By adjusting the biases, a more structured method could be introduced to directly optimise the overall generalisation performance of the multi-class classifiers.

ments still in progress.

[20]The solutions simply restricted the size of *working set* (i.e. number of support vectors) and this resulted in poor solutions.

Chapter 4

Probability outputs

Almost all speech recognition systems are based on statistical models ([30]). If the proposed multi-class system 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 since the output of RNNs can be interpreted as posterior probabilities ([2]). 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 [18] and [47] there is no clear relationship between this distance and the posterior class probabilities. If we can find such a relationship, it will be possible to generate N-best lists[1] of 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 seem logically more likely to be correct. A very rough estimate of posterior probability could be obtained by scaling down the number by $1/780$ such that the sum of votes are 1. These values could be interpreted

[1] See [49].

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 [60]. 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). [47] 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 SVM 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. [47] 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 defined 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 [47] is to use a separate validation set. This is easy in this case, since I already use a validation set to learn the SVM parameters.

Figure 4.1 is taken from [47], 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 fairly good estimate of the probability. This result is consistent with results reported on other speech data sets ([18]).

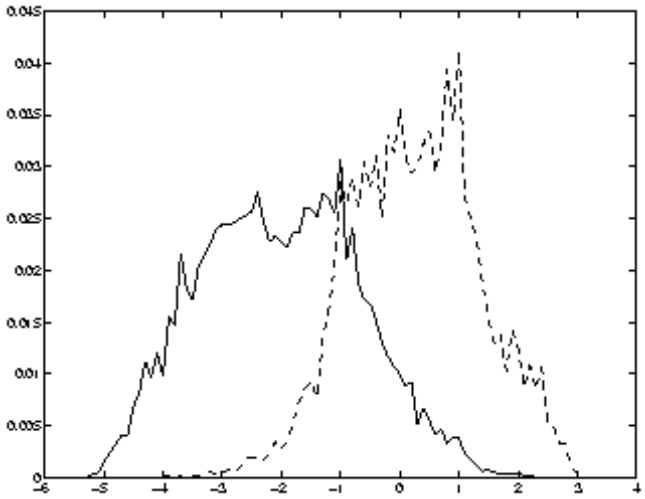

Figure 4.1: Histogram of posterior probabilities for a model trained using a linear kernel (from [47]). Note the non-Gaussian distributions.

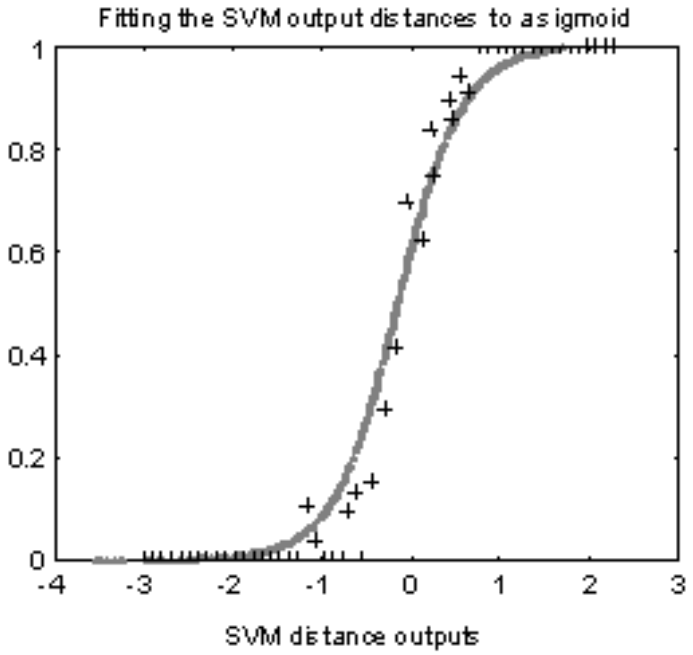

Figure 4.2: A sigmoid fit to the distance-based posterior probability estimates (/aa/ vs. /ae/).

4.2 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 result in fairly accurate estimates, and thus makes it possible to use SVMs as one of the basic units in a speech recognition system.

Chapter 5

Conclusion and future work

The goal of this thesis was to examine the feasibility of applying Support Vector Machines to the problem of frame classification. This was done, and the SVM method was found to create a capable learning machine with good generalisation ability. 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 counter act the *curse of dimensionality* and generalise well even when the data contained numerous irrelevant features.

In chapter 3 I described three 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% are competitive with the best results found in literature, even under sub-optimal conditions. 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. Lastly, 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 fairly accurate 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.

However, all hurdles in making SVMs a viable options in real-time speech recognisers were not solved. 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. The best reported result was found by the DAGSVM classifier. This classifier needed to evaluate 39 binary SVMs before it returned a prediction. This required far more time than what is allowed in a real-time system. Fortunately, several ways to reduce classification time exist.

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[1]. The binary tree classifier described made use of such groupings. It reduced the number of evaluations to 3-10. However, the performance of this classifier was not competitive since it was forced to limit the amount of training data in the top nodes of the tree.

2. *Data-cleaning*. As described in section 2.4.2, the solution of the SVM training algorithm is typically sparse. However, if the data sets have a large amount of overlap as is common among the speech data, the solution will be less sparse. In other words, the resulting model will contain a large amount of support vectors, and consequently classification time will be high. In most cases, this is quite unnecessary. [18] and [29] observed that a sparse solution with similar generalisation performance could be found by throwing away all support vectors not lying on the decision boundary. Moreover, they note that even in sparse solutions this technique lead to drastic reductions in support vectors, and hence classification time.

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

An mentioned above, a key to better generalisation performance is to solve the problem of how training time scales with the traning set size. However, a solution is not straightforward. Not only should it be able to promptly handle large amounts of training data, but it also needs to consider memory requirements and naturally, produce optimal solutions. Such a method must be discovered to unlock the full potential of SVMs in speech recognition.

Appendix A

Kernel Requirements

The following theorems are taken from [4].

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.

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

with:

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

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

Binary Tree Structure

On figure C.1, the binary tree structure grown by AddTree can be seen.

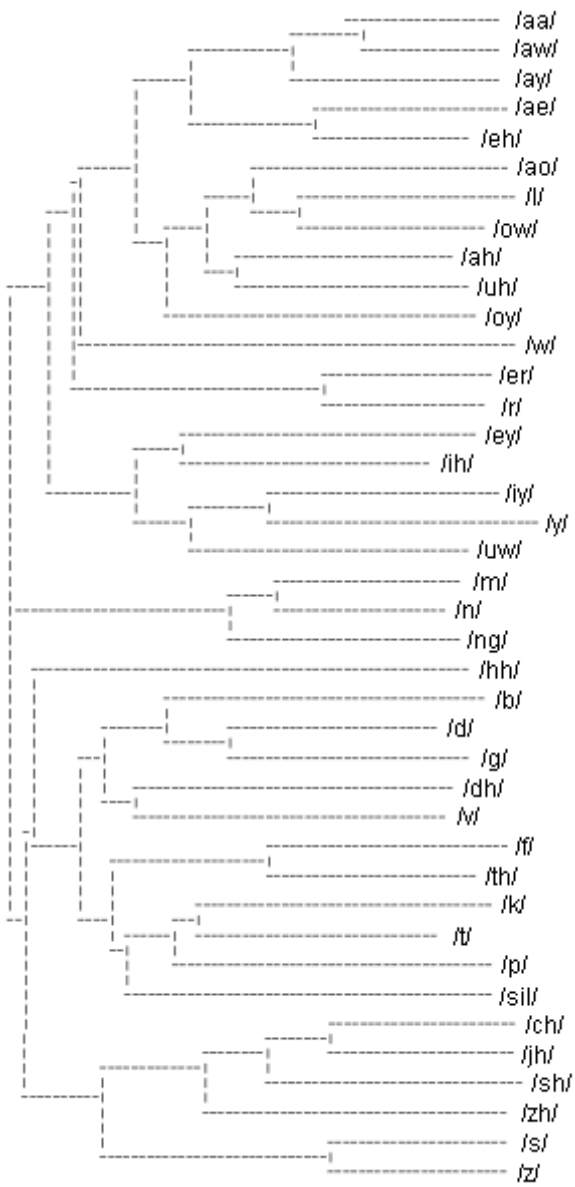

Figure C.1: Tree structure grown using AddTree algorithm.

Bibliography

[1] 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.

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

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

[4] 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.

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

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

[7] N. Chomsky and M. Halle. *The Sound Patterns of English*. MIT Press, Cambridge, MA, USA, 1968.

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

[9] 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.

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

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

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

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

[14] J. Corter. Addtree/p: A pascal program for fitting additive trees based on sattath and tversky's addtree algorithm. In *Behavior Research Methods and Instrumentation, 14(3)*, pages 353–354, 1982.

[15] 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.

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

[17] G. Fung and O. Mangasarian. Proximal support vector machine classifiers. In F. P. D. Lee and R. Srikant, editors, *KDD2001: Knowledge Discovery and Data Mining*, pages 64–70, New York, 2000.

[18] A. Ganapathiraju. *Support Vector Machines for Speech Recognition*. PhD thesis, Mississippi State University, US, 2001.

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

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

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

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

[23] 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.

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

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

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

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

[28] 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.

[29] 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.

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

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

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

[33] B. Leclerc. Consensus of classifications: the case of trees. In *IFCS-98*. Springer-Verlag, 1998.

[34] 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.

[35] 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.

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

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

[38] 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.

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

[40] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

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

[42] 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.

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

[44] 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.

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

[46] 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.

[47] 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.

[48] J. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

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

[50] 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.

[51] S. Sattath and A. Tversky. Additive similarity trees. In *Psychometrika 42*, pages 319–345, 1977.

[52] M. Schmidt and H. Gish. Speaker identification via support vector classifiers. In *ICASSP, volume 1*, 1996.

[53] B. Schölkopf. *Support Vector Learning*. R. Oldenbourg Verlag, Munich, 1997. To order click here or here.

[54] 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.

[55] 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.

[56] A. J. Smola. Regression estimation with support vector learning machines. Master's thesis, Technische Universität München, 1996.

[57] 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.

[58] D. Tax and R. Duin. Data domain description by support vector. In *ESANN99*, pages 251–256, 1999.

[59] R. Vanderbei. Loqo: An interior point cpde for quadratic programming. Technical report, Priceton University, US, 1994.

[60] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, N.Y., 1995.

[61] 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.

[62] 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.

[63] C. Williams. Support vector machines (course notes),university of edinburgh, uk, 2000.