Speaker Verification
using
Support Vector Machines

Vincent Wan

Department of Computer Science
University of Sheffield
United Kingdom

June 2003

Dissertation submitted to the University of Sheffield
for the degree of Doctor of Philosophy
# Contents

List of Figures ......................................................... iv  
List of Tables .......................................................... vi

Abstract vii

Acknowledgments viii

1 Introduction 1

2 Speaker verification 4  
  2.1 Introduction .................................................. 4  
  2.2 Extracting speaker specific features ............... 6  
    2.2.1 Frequency analysis .................................. 6  
  2.3 Decision making ............................................. 9  
    2.3.1 Hypothesis testing ................................... 9  
    2.3.2 Error reporting ......................................... 10  
  2.4 Corpora ......................................................... 10  
    2.4.1 The YOHO voice verification corpus ............ 12  
    2.4.2 The PolyVar database ................................ 12

3 Creating models of speakers 14  
  3.1 Modelling strategy ........................................... 14  
    3.1.1 Discriminative vs. generative models ......... 14  
  3.2 Generative models ........................................... 17  
    3.2.1 Vector quantisation .................................... 17  
    3.2.2 Gaussian mixture models ....................... 17  
    3.2.3 Hidden Markov models ............................. 19  
  3.3 Discriminative models ..................................... 24  
    3.3.1 Support vector machines ....................... 24  
    3.3.2 Polynomial classifiers .......................... 24  
    3.3.3 Neural networks ................................. 26  
  3.4 Making generative models discriminative .......... 27  
    3.4.1 Radial basis function networks ......... 27  
    3.4.2 GMM-LR/SVM combination .................. 28  
    3.4.3 HMM/MLP hybrids ......................... 29  
    3.4.4 Discriminative training of HMMs .......... 30
4 Support vector machines ........................................ 34

4.1 Introduction .................................................. 34
4.2 Machine learning by risk minimisation ......................... 34
4.3 Binary linear decision boundaries .............................. 35
  4.3.1 Maximising the margin ................................ 36
  4.3.2 Support vectors — memorising the training data ........ 38
  4.3.3 The non-separable case ............................... 38
  4.3.4 Choice of cost function ................................ 40
4.4 Non-linear decision boundaries ............................... 41
  4.4.1 Mapping to high dimensions ............................ 41
  4.4.2 From dot products to kernels ......................... 43
  4.4.3 General kernels ..................................... 44
4.5 Classifying sequences using score-space kernels .......... 44
  4.5.1 Defining the score-space .............................. 45
  4.5.2 Computing the score-space vectors ................... 47
  4.5.3 Dot products in score-space ......................... 48
4.6 Optimising SVMs ............................................. 49
4.7 Properties of SVMs .......................................... 50
  4.7.1 Regularised risk ...................................... 50
  4.7.2 Comparison with RBFs .................................. 50
  4.7.3 SVM extensions for more than two classes ............ 51
4.8 Concluding remarks ........................................ 54

5 Developing speaker verification SVMs on YOHO .......... 55

5.1 Introduction .................................................. 55
  5.1.1 Layout of this chapter ................................ 56
  5.1.2 Experimental procedure .............................. 56
5.2 SVMs vs. polynomial classifiers ............................ 57
5.3 Spherically normalised polynomial kernels .................. 60
  5.3.1 RBF interpretation ................................... 64
  5.3.2 Properties of the spherically normalised general polynomial kernel ................................. 64
  5.3.3 Spherical normalisation for any kernel ............... 65
  5.3.4 Experiments ......................................... 67
5.4 Score-space kernels ......................................... 68
  5.4.1 The baseline GMM ................................. 69
  5.4.2 Experiments ......................................... 69
5.5 Conclusion ..................................................... 70
  5.5.1 The polynomial sequence kernel .................... 71

6 Evaluation on PolyVar ....................................... 72

6.1 Introduction .................................................. 72
  6.1.1 PolyVar protocol for training and testing .......... 73
6.2 The GMM baselines .......................................... 73
6.3 SVM evaluations ....................................................... 74
  6.3.1 Polynomial kernels ............................................. 74
  6.3.2 Score-space kernels ........................................... 82
  6.3.3 Overall ranking ................................................ 84
6.4 Conclusion .......................................................... 85

7 Conclusion .............................................................. 87
  7.1 Summary .......................................................... 87
  7.2 Future work: spherical normalisation .......................... 89
  7.3 Future work: Pair HMMs .......................................... 89
    7.3.1 Introduction .................................................. 89
    7.3.2 The pair-wise alignment model .............................. 90
    7.3.3 The forward and backward recursion ...................... 92
    7.3.4 The pair HMM as a kernel function ....................... 93
    7.3.5 Implementation .............................................. 96
    7.3.6 Preliminary result and discussion ........................ 98

Bibliography .............................................................. 100

Index .............................................................................. 107
List of Figures

2.1 A generic speaker verification system ................................................. 5
2.2 Outline of the steps taken in perceptual linear predictive analysis of speech [Her90] .... 8
2.3 Reporting classifier performance on ROC and DET curves ....................... 11
3.1 The difference between generative and discriminative modelling approaches. .......... 16
3.2 Finite state generator HMM .............................................................. 20
3.3 Illustration of the forward recursion alignment matrix .............................. 22
3.4 A multilayer perceptron ....................................................................... 26
3.5 A general radial basis function network. .............................................. 28
3.6 A GMM/SVM hybrid classifier. ............................................................ 29
3.7 The relationships between discriminative and generative models ................. 33
4.1 Graph of risk vs. complexity. ................................................................. 35
4.2 How a support vector machine works on separable data. (Source [Bur98].) .......... 37
4.3 The soft margin: adapting the SVM to work with non-separable data. (Source [Bur98].) 39
4.4 A linear decision in a high dimensional space corresponds to a non-linear decision in the input space. ........................................................... 42
4.5 Visualising dot products in input space and feature space. .......................... 43
4.6 A radial basis function network with one output. .................................... 51
4.7 A simple comparison between RBF networks and an SVM with the RBF kernel. .... 52
4.8 Directed acyclic graph for classifying four classes using binary classifiers ........... 53
5.1 Illustration of how the maximum margin criterion is not invariant to linear transformations. 61
5.2 Spherical vector-length normalisation: mapping onto a sphere .................... 62
5.3 Spherical vector-length normalisation: various ways of projecting data onto unit hyperspheres. ................................................................. 63
5.4 Plots of the spherically normalised general polynomial kernel output scores as a function of the angle between vectors illustrate the effect of choosing different values of the kernel parameters. .................................................. 66
6.1 DETs of the baseline classifiers on PolyVar .......................................... 76
6.2 DETs of the polynomial kernel SVMs on PolyVar .................................. 77
6.3 DETs of the Fisher kernel SVMs on PolyVar ...................................... 78
6.4 DETs of the likelihood ratio score-space kernel SVMs on PolyVar .............. 79
6.5 Comparison of all approaches on PolyVar ........................................... 80
List of Figures

7.1 The pair hidden Markov model ............................................ 91
7.2 The pair HMM state substitution and notation ......................... 95
List of Tables

4.1 A short list of some basic kernel functions [SSW+98] ........................................ 45
4.2 Some examples of score operators ................................................................. 45
4.3 An example of error correcting codes ......................................................... 53
5.1 SVM vs. polynomial classifiers: summary of results. The polynomial classifier results
were kindly provided by Campbell [CTB00]. SVMs were trained, each time modifying a
different property to determine its effect on the classifier’s performance. The test reported
figures are the average test EERs over all the speakers in the database. .................. 58
5.2 Size of feature space as a function of polynomial order .................................. 59
5.3 Error rates of spherically normalised polynomial kernels trained on YOHO ........ 67
5.4 GMM baseline performance on YOHO. ....................................................... 69
5.5 Performance of Fisher kernel SVMs on text independent speaker verification and speaker
identification tasks. ............................................................................................ 70
5.6 Detailed breakdown of Fisher kernels on text independent speaker verification .... 70
6.1 Results of the PolyVar experiments ............................................................... 75
6.2 Effect of vector quantisation on classification performance of a fifth order spherically nor-
malised polynomial kernel SVM. The quantisation of the background speakers was kept
constant at 1500 codebook vectors while the number of client codebook vectors was varied. 81
6.3 The effect of the likelihood score on score-space classifiers .............................. 84
Abstract

Current state-of-the-art speaker verification systems are based on discriminatively trained generative models. In these systems, discrimination is achieved at the frame-level: each frame of data is scored separately and then the scores are combined to compute the score for the whole utterance. A better way is to apply discrimination on the sequence as a whole rather than on its constituent parts.

In this thesis we develop the techniques required for SVMs to work well on speaker verification using the sequence discrimination approach. The main focus of attention is on the kernel function. We begin with an investigation into the polynomial kernel for frame-level discrimination before moving to the Fisher and score-space kernels to achieve sequence-level discrimination.

The polynomial kernel is less widely used than, for example, the radial basis function kernel. We examine the properties of the polynomial kernel in relation to a polynomial classifier. In doing so we develop a new technique called spherical normalisation that can improve the performance of the polynomial kernel. Spherical normalisation may be seen as a preconditioning step that is completely general and not just restricted to polynomial kernels. We also applied the technique to the score-space kernel functions and found that spherical normalisation benefits these kernels also. Spherical normalisation has some interesting properties including the ability to impart upon any valid kernel properties of radial basis function kernels. Spherical normalisation is a valuable tool for researchers wanting to develop new types of SVM kernel: in some preliminary experiments it enabled pair HMM kernel SVMs to be trained much more easily.

For whole sequence discriminative classification we study the set of score-space kernels, which includes the Fisher kernel, for deriving non-linear transformations from a variable length sequence to a fixed length vector. These kernels exploit generative models to achieve the non-linear mapping. By representing the entire sequence as a single vector, the SVM can discriminate between whole sequences directly. In contrast a frame-level discriminative classifier may discard information not useful for frame discrimination but useful for sequence discrimination. Experimentally, a support vector machine combined with a score-space kernel and spherical normalisation can out perform current state-of-the-art classifiers on the PolyVar speaker verification database. This thesis reports equal error rates on the PolyVar database that are 34% lower than the baseline Gaussian mixture model likelihood ratio approach.
Acknowledgments

Firstly, I would like to thank my supervisor, Steve Renals, for his support, advice and guidance in matters of practical, theoretical and financial nature throughout this PhD. I would also like to thank all the members of the Speech and Hearing Group in the Department of Computer Science for their support and discussions.

I gratefully acknowledge the financial support provided by Motorola. I would like to thank everyone at Motorola Human Interface Lab, Palo Alto, with special thanks to Sreeram Balakrishnan, Steve Nowlan, Eddy Mayoraz, Sadik Kapadia, Tasos Anastasakos, Giovanni Seni and Michael Van Wie for our many interesting discussions. Thanks also to William Campbell for furnishing me with the YOHO database and his polynomial classifier results reported herein.

Thanks to Johnny Mariëthos of IDIAP for his advice in training the baseline GMMs on the PolyVar database. Special thanks to Brian Mitchell for helping me with latex, for proof reading this document and many other things. Finally, thanks to my father and brother for their love and support and to Sook Ching for her patience.
Chapter 1

Introduction

With the growth of modern telecommunications and multimedia systems and the availability of cheaper and faster processors comes an increasing need for technologies that cope with speech reliably. Speech recognition involves determining the linguistic content of an utterance. In contrast, speaker recognition is concerned with identifying who spoke an utterance by using the person’s voice as a biometric measurement to determine their identity uniquely. Applications of speaker recognition systems range from computerised access control via telephone to speaker-dependent audio indexing in multi-speaker environments. The focus of this thesis is speaker verification, which is concerned with authenticating a claimed identity, in access control applications for example.

In speech recognition the aim of the classifier is to determine the probabilities of a set of sub-word units for processing by a language model later. Speaker verification is concerned solely with the validity of the speaker’s claimed identity. Speaker verification is a direct binary classification problem that does not require much post-processing: only hypothesis testing to adjust the trade-off between the relative number of errors the classifier makes. Current state-of-the-art speaker verification systems use generative model classifiers such as Gaussian mixture models (GMMs) and hidden Markov models (HMMs) (see section 3.5.1).

If the true model is not known then generative models are not the best approach to use for classification (see chapter 3) since their goal is to model the separate classes as best they can, not to make as few classification errors as possible. It is the goal of discriminative models to minimise the number of classification errors. However, researchers in speaker verification have reported better results using generative models. Simple discrimination has been imparted to generative verification systems by taking the ratio of the client model to a world model (a model of background speakers), thereby increasing the separation between client and potential impostors (see section 3.2.2). This is known to improve classifier performance in speaker verification tasks. Researchers have reported improvements over a generative model by incorporating discrimination after the generative models are trained [BM01]. There are other, more principled, approaches to discriminative training of generative models such as training by maximising the mutual information between the model and the utterance. A review of such approaches is provided in chapter 3. A discriminative approach that is particularly suited to speaker verification comes in the form of the support vector machine (SVM).

The support vector machine (SVM) [Vap95] is a discriminative approach that seems well suited to speaker verification. It was developed in the early to mid 1990s and quickly became widely known. Initial speaker verification results using SVMs reported by Schmidt and Gish [SG94] were promising. Its novel maximum margin criterion immediately sparked interest and led to a powerful classifier that is interesting
theoretically and experimentally. It is a classifier that generalises well on small quantities of data and does not suffer from the curse of dimensionality as greatly as some others (see chapter 4). Much research has been done to apply SVMs to different fields including handwriting recognition, natural language processing and speech processing.

The SVM is a nonparametric classifier. It is so named because it defines the class decision boundary using a subset of the training data, called the support vectors, that are nearest the boundary and hence most difficult to classify. The support vectors are found by optimising a convex objective function for which a globally optimum solution may always be found. The solution’s support vectors contain the subset of training samples from the set of background speakers that are closest to the client. Some speaker verification classifiers (see section 3.2.2) use a separate processing stage to find these speakers. By focusing more on the speakers that are most similar to the client, the resulting classifier is better because it is been trained on data that are more difficult to classify. The SVM approach to speaker verification may be viewed as an approach that combines discrimination (classification error minimisation) with background speaker selection in a single optimisation step.

There is another benefit that the SVM has. In an utterance of speech supplied by a speaker there is a large amount of correlation between frames or segments within the utterance. Even if the utterance transcription (i.e. the actual ordering of the frames) is ignored there still exists some correlation. The fact remains that the frames are all spoken by one speaker and are part of the same utterance. Most existing classification techniques ignore this fact. They are intrinsically incapable of classifying sequences directly so simplifying assumptions must be made. Most commonly, the frames are assumed to be independent of each other. This allows classifiers to score each frame separately and the utterance classification is inferred from the sequence of scores (often by computing their mean). By scoring frames independently, correlations between frames are easily lost. Discriminative classifiers (described in chapter 3) discard information that is deemed not useful for classification. Therefore, when scoring frames independently, such classifiers may discard information that would have been useful for classifying the complete utterance. In contrast generative models try to model everything. This may be the reason that discriminative methods have not done as well as generative methods. Herein lies the benefit of the SVM. By exploiting the generative model it is possible to map a variable length utterance to a representation of fixed length which can be classified later (a generative model is used since it estimates the probability densities of the data and lessens the information lost in the mapping). Performing such a mapping to a single vector hopefully retains the information useful for sequence discrimination previously discarded at the frame-level. The fixed length representation is achieved by Fisher [JH99] and score-space kernels [SGN01] and can have a very high dimensionality so SVMs are best suited to classify them: other methods are subject to problems such as parameterisation and over-training.

Speaker verification requires a binary decision to be made. Since SVMs are discriminative binary classifiers that can classify a complete utterance in a single step whilst simultaneously finding the most difficult to classify background speakers then they should be well suited to the task. Speaker verification may also be seen as a stepping stone to the application of SVMs to other speech related tasks including speech recognition. The nature of the speaker verification task enables us to focus on the properties of SVM without needing to contend with the effects of combining binary decisions to solve problems that involve more than two classes or the difficulties in obtaining posterior probability estimates.

We examine a number of different approaches to speaker verification using SVMs. Our study starts by treating the SVM as a static classifier and uses a separate averaging step to cope with variable length sequences (chapter 5). In particular we compare a polynomial classifier to a polynomial kernel SVM. The comparison leads to a new normalisation method that improves SVM performance and is used throughout
the thesis. Then we move to classifying whole utterances by using an GMM as a pre-processing step prior to SVM classification. In our evaluations we find that discrimination at the sequence level is highly effective for speaker verification and performs better than current state-of-the-art classifiers.

**Thesis layout**

Chapter 2 provides a review of speaker verification including feature extraction, hypothesis testing and error reporting. It also describes the two databases used in this thesis: the YOHO database used for development work in chapters 5 and the PolyVar database used for the final evaluation in chapter 6.

Chapter 3 describes the modelling approaches that may be used to model speakers (or speech in general). Discriminative and generative modelling approaches are described and some classifiers that combine the two are reviewed. The combination of discriminative and generative approaches provides some of the motivation for the work in chapter 5. Chapter 4 describes the fundamentals of SVMs for pattern classification in detail and provides a literature review of the development research that has been done.

In chapter 5 we examine the properties of SVMs by comparing a polynomial kernel SVM with a polynomial classifier. The polynomial kernel is less widely used than, for example, the radial basis function kernel. This is a result of the nature of polynomials, where the magnitudes of their output increase rapidly at extreme values making polynomial SVMs harder to train. In our case, we have a baseline polynomial classifier to compare against: a classifier that is known to do well on the task. In making the comparison we develop the spherical normalisation technique that can improve the performance of the polynomial kernel to match the polynomial classifier. Spherical normalisation is a preconditioning step that is applied like a kernel and has some interesting properties such as an RBF interpretation. It is a completely general technique and not restricted to polynomial kernels. Thus when applied to other kernels to precondition the optimisation problem RBF properties are imparted upon those also. We also applied the technique to some sequence kernels and found that spherical normalisation benefits them also.

The current state-of-the-art classifiers in speech processing are based on discriminatively trained generative models. This provides a good motivation to apply sequence kernels to speaker verification. Sequence kernels are derived from generative models and the support vector machine provides the discrimination. Also in chapter 5 we study the set of score-space kernels, specifically the Fisher kernel, for deriving non-linear transformations from any parametric generative model.

In chapter 6 we evaluate the methods on a different dataset to verify the results from the development phase. We show that, experimentally, an SVM combined with a score-space kernel and spherical normalisation can achieve state-of-the-art results. Finally, chapter 7 concludes the thesis and includes a brief outline of some preliminary work on a different sequence kernel: one derived from the pair hidden Markov model with conditional symmetric independence constraints.
Chapter 2

Speaker verification

2.1 Introduction

In speech processing, recognition may refer to speech recognition, which determines the linguistic content of an utterance, or speaker recognition, which determines the talker’s identity. Speaker recognition may be further divided into speaker identification and speaker verification. The aim of a speaker identification system is to identify the person who spoke the utterance from a pool of possible speakers or to identify the speaker as unknown. The focus of this thesis is speaker verification, which is used to authenticate a person’s claimed identity.

The speaker verification procedure, as far as a user of such a system is concerned, is similar to that of a password entry system. The user, from here to be known as the claimant, claims the identity of a client, someone authorised to access the system. The verification system requests the claimant give a sample of speech into a microphone. The speech may be prompted or it may be a predefined verification phrase. The system then processes the recorded speech and compares it to a model of the client’s voice stored in its database. If it matches then the claimant is authenticated. The length of the verification phrase may affect the system accuracy. Usually, longer verification phrases yield more accurate systems. Depending upon the type of security, the system may request another sample of speech if the match is borderline. For extra security, the system may also pass the recorded speech to a speech recogniser to ensure that the correct response is given. In a system where speech is prompted, this extra security measure prevents the use of recordings.

Speaker verification may be approached as a text dependent or text independent problem. In a text independent system a specific verification phrase is not required nor is a transcription of the phrase. A text independent system is capable of authenticating claimants independently of what is spoken. This approach, therefore, models only the sound of the client’s voice, which is determined by the physical characteristics of the client’s vocal tract. The text dependent approach requires a specific phrase or a transcription of the phrase. When the phrase (or transcription) is known then it is also possible for the system to model learned characteristics of the client’s voice, such as speaking rate and accent. Since a text dependent system models more variability in a person’s voice then it is generally more accurate than a text independent system. Text independent speaker verification is considered more difficult since linguistic variability must be ignored and thus accuracy is lower.

During the authentication procedure, the verification system compares the claimant’s speech with the client model. The creation of the client model is known as enrolment. A system that has more data available
2.1. Introduction

Figure 2.1: A generic speaker verification system

to it is capable of creating more accurate client models and so has greater accuracy. Therefore, to enrol onto a speaker verification system, a client should supply as many samples of speech as possible. Enrolment is complicated by the fact that there may be a large amount of variability in a person’s voice. Variabilities arise depending upon the state of the vocal tract, which is affected by influences both internal (such as a cold or a dry mouth) and external (such as temperature and humidity). Only by taking samples of the client’s voice at regular intervals over a long period of time may the model capture many of the different states that the vocal tract may take. However, depending upon the application there is a practical limitation on the amount of data that each person is able or willing to provide. A system that operates over a public telephone network would ideally enrol a client with only a few seconds of speech. The quantity of data used to build a client model directly affect the accuracy of a system. There are many other uncontrollable factors that limit the performance of speaker verification systems.

The performance of a speaker verification system is also affected by room and line noise. In a controlled office environment noise levels can be minimised relatively easily. If speakers are co-operative then the recordings will be high quality and the overall system performance will be greater. At the other end of the scale, such as over-the-phone applications, there may be background noise, variability in the recording equipment and the claimant may be impatient and uncooperative. These are a few of the important factors that cannot be ignored and ultimately limit the accuracy of any system. In this thesis we shall be looking at improving the performance of speaker verification systems by extracting more information from existing client models or by building better models.

Automatic speaker verification consists of a number of distinct steps: speech data acquisition, feature extraction, pattern matching, decision making and enrolment. The sequence of events is shown in figure 2.1. The first step, speech acquisition, involves recording a sample of the client’s voice. The feature extraction step converts the acquired waveform into a sequence of vectors or frames. Each frame represents a short window of the waveform, typically around 30 milliseconds, with overlapping adjacent windows. The enrolment phase creates a model for each of the clients’ voices using samples of their speech. The
pattern matching step, for authenticating claimants, scores the test speech using the model of the client created during enrolment. Various modelling strategies and pattern matching techniques are discussed in chapter 3. Lastly, the scores are combined and hypothesis testing is used to determine whether to accept or reject the claimant.

This chapter reviews feature extraction, decision theory and error reporting in speaker verification. It also introduces the two databases that are used to test the speaker verification systems in this thesis. Finally, a brief overview of the relative performance of different modelling and pattern matching techniques is given.

2.2 Extracting speaker specific features

The production of speech is a complicated process. There are two main sources of inter-speaker variability or speaker specific characteristics: physical characteristics and learned characteristics. Physical characteristics are determined by the shape and size of the vocal tract, which consists of the vocal cords and the organs and spaces that lie above them. These include:

1. the laryngeal pharynx (the area between the vocal cords and the epiglottis).
2. the oral pharynx (from the epiglottis to the back of the tongue and the velum or soft palate).
3. the oral cavity (the area in front of the velum bounded by the lips, the tongue and the palate).
4. the nasal pharynx (above the velum, the rear end of the nasal cavity).
5. the nasal cavity (above the palate, extending to the nostrils).

Acoustic waves, created by air rushing past the vocal cords, travel through the vocal tract. The frequencies of the waves are altered by the resonances of the vocal tract.

A text independent speaker recognition system typically uses features derived from a frequency analysis of the voice to infer the shape of the vocal tract thereby identifying the speaker. Learned characteristics of a voice include speaking rate, dialect and prosodic effects. Text dependent speaker recognition systems may try to capture these effects to improve speaker discrimination.

2.2.1 Frequency analysis

Building models of speakers depends upon a frequency analysis of the speaker’s voice. The data rate of a waveform is extremely high. Performing a frequency analysis provides information about the spectral content of the waveform at a significantly lower data rate. There are many standard methods that can be used.

Linear predictive coding

A commonly used method is linear prediction [Mak75] or linear predictive coding (LPC). LPC assumes that speech can be modelled as the output of a linear, time-varying system excited by either periodic pulses (to model voiced speech, which is generated by the vibration of the vocal cords) or random noise (to model unvoiced speech, generated when the vocal cords are relaxed).
LPC performs spectral analysis on windows of speech using an all-pole modelling constraint. Within each window the sample at time \( t \), \( s(t) \) is assumed to be expressible as a linear combination of the past \( p \) samples plus some excitation term, \( Gu(t) \):

\[
s(t) \approx \sum_{i=1}^{p} a_i s(t-i) + Gu(t)
\]

where \( a_i \) are assumed to be constant over the window of speech, \( G \) is the gain and \( u(t) \) is the normalised excitation. Expressing equation (2.1) in the \( z \)-domain

\[
S(z) = \sum_{i=1}^{p} a_i z^{-i} S(z) + GU(z),
\]

where \( S \) is the \( z \) transform of \( s \) and likewise for \( U \). Rearranging (2.2) gives

\[
H(z) = \frac{S(z)}{GU(z)} = \frac{1}{1-\sum_{i=1}^{p} a_i z^{-i}}
\]

which corresponds to the transfer function of a digital time-varying all-pole filter.

The parameter, \( p \), is called the LPC analysis order. The output of the LPC analysis on a window of speech is the vector of \( p \) coefficients (known as the autoregressive coefficients), \( a_1 \ldots a_p \), that specify the spectrum of an all-pole model that best matches the spectrum of frequencies inside the window. The solution for these LPC coefficients is obtained by minimising the mean squared error between the model and the signal and solving the resultant set of \( p \) simultaneous equations. The autoregressive coefficients can be further transformed into the cepstral coefficients of the all-pole model.

### Perceptual linear prediction

Perceptual linear prediction (PLP) [Her90] combines LPC analysis with psychophysics knowledge of the human auditory system. With respect to LPC, PLP analysis applies three transformations to the speech signal to simulate the perceptual properties of the human ear. These transformations are applied prior to building the all-pole model.

An outline of the steps in PLP analysis is shown in figure 2.2. The three psychophysics based transformations are critical band analysis, equal-loudness pre-emphasis and intensity loudness conversion. Critical band analysis simulates the non-uniform frequency resolution of the auditory system: the human ear has a higher frequency resolution at low frequencies than it does at high frequencies. This is achieved by mapping the frequency scale onto the Bark scale. Equal-loudness pre-emphasis simulates the non-equal sensitivity of hearing at different frequencies. Finally, intensity loudness conversion simulates the non-linear relationship between the amplitude of a sound and its perceived loudness using a cube root amplitude compression.

### Additional post-processing

For systems that operate on variable communication channels such as public telephone networks the extracted features must be corrected for channel variations. A simple method to achieve this is called cepstral mean subtraction. The means of the features are estimated over a section of speech. Each feature has this mean subtracted away thereby removing any linear, time invariant static noise from the features [SCI75].

After the cepstral coefficients have been obtained extra information can be derived from them. The first and second derivatives (with respect to time) of the cepstral coefficients give an indication of the rates at which the cepstral coefficients change. It has been found that augmenting the cepstral coefficients with derivatives often improves the accuracy of speech recognition and speaker recognition systems.
2.2. Extracting speaker specific features

Figure 2.2: Outline of the steps taken in perceptual linear predictive analysis of speech [Her90].
2.3 Decision making

The pattern matching algorithms of chapters 3 and 4 compute a score indicating how well the verification phrase matches the client’s model. Let $X = \{x_1 \ldots x_N\}$ denote an utterance consisting of a sequence of $N$ frames. Each frame, $x_i$, corresponds to one vector of LPC or PLP coefficients described earlier. Let $S(X)$ denote the score assigned to the whole utterance by the pattern matching algorithm or classifier. It is from these utterance scores that a decision is made. To decide whether to authenticate a claimant, simply compare $S(X)$ to a threshold, $T$. If $S(X)$ is greater than $T$ then authenticate the claimant else reject him. This process is formally known as hypothesis testing.

2.3.1 Hypothesis testing

Speaker verification is a binary classification problem and involves choosing between two possible outcomes. Let $H_0$ denote the hypothesis that the claimant is an impostor and $H_1$ denote the hypothesis that the claimant is the client. In hypothesis testing there are four possible outcomes. Two outcomes are when the hypotheses are correct: $H_0$ chosen when the claimant is an impostor, $H_1$ chosen when the client is the client. The remaining outcomes are errors. A type I error occurs when an impostor is authenticated (a false-acceptance, FA). A type II error occurs when the client is rejected (a false-rejection, FR).

The utterance scores of a client model have two different, usually overlapping, probability density functions. One pdf represents the client’s scores and the other represents impostors’. We shall denote these conditional probability density functions by $p(X|H_0)$, that the verification phrase $X$ was from an impostor, and $p(X|H_1)$, that the phrase was from the client. In statistical decision theory the choice between the two hypotheses is made based upon the posterior probabilities of each. If the posterior probability $p(H_0|X)$ is the greater then choose $H_0$ else choose $H_1$. Assuming that the cost of making type I and II errors are equal then, using Bayes rule, the ratio of the posteriors is,

$$
S(X) = \frac{p(H_1|X)}{p(H_0|X)} = \frac{p(X|H_1)}{p(X|H_0)}. \tag{2.4}
$$

Written in this way if $S(X) \geq 1$ then choose $H_1$ else choose $H_0$. Often the conditional probability distributions and posterior probabilities are not known. In such cases the conditional probability densities can be estimated from the training data. An estimate of $p(X|H_1)$ can be made by computing a sample of utterance scores of the client. Likewise, an estimate of $p(X|H_0)$ can be made on a sample of utterance scores from impostors.

The decision rule can be generalised to a Neyman-Pearson type rule or to the Bayes decision rule with a cost function [Fuk90] by replacing the constant threshold with a variable one. Thus the decision rule is,

$$
\text{if } S(X) \geq T, \text{ choose } H_1 \\
\text{if } S(X) < T, \text{ choose } H_0 \tag{2.5}
$$

For speaker recognition tasks it is important to remove frames that represent silence since they contain no information about the speaker. Once the processing is complete the sequences cepstral coefficients are used as input to the modelling methods described in chapters 3 and 4.
The threshold $T$ may be set

1. so that it approximately minimises the number of errors made by the system.
2. so that the rate of either false-accepts or false-rejects is at a fixed value.
3. so that a desired false-accepts to false-rejects ratio is achieved.

### 2.3.2 Error reporting

The threshold may be varied depending upon the application, for example, in high risk applications $T$ may be set so that the chances of a false-acceptance occurring is low. In speaker verification a common statistic to quote is the equal error rate (EER). This is the error rate that occurs when the threshold is set such that the rate of false-accepts is equal to the rate of false-rejects.

Another commonly quoted statistic is known as the half total error rate (HTER). It is used when no prior information on the cost of different kinds of errors is known. The HTER is the arithmetic mean of the false acceptance rate and the false rejection rate at a given threshold.

\[
HTER = \frac{1}{2} (\%FA + \%FR) \]  

Typically the threshold is adjusted so as to minimise the HTER. The HTER changes the relative weight of client and impostor accesses in order to give them equal weight instead of the priors induced by the data.

The EER and HTER define two particular trade-offs between false acceptances and false rejections. A better picture of the performance of a classifier may be seen on a receiver operating characteristic (ROC) curve [Swe64, Ega75], which illustrates the trade off between the two error types. Generally the false acceptance rate is plotted on the horizontal axis and the correct detection rate (equivalent to one minus the false rejection rate) is plotted on the vertical axis. Figure 2.3a shows an example ROC curve. The area under the curve gives an indication of how well the classifier performs: the closer the area is to one the better the classifier.

An ROC curve may be transformed to a detection error trade-off (DET) curve [MDK+97] shown in figure 2.3b. The false acceptance rate is plotted on the horizontal axis and the false rejection rate on the vertical axis. The difference is the scale on the axes: the curves are plotted using the normal deviate scale, where the percentage rates are expressed as the number of standard deviations from the mean of a normal distribution. Figure 2.3c illustrates how the percentage area under the normal distribution corresponds to the number of standard deviations. On this scale the trade-off curve is more linear and differences between different types of classifier are easier to see.

It is straight-forward to plot a detection error trade-off curve for a classifier trained for a single speaker (simply compare the list of utterance scores generated from a test set to a threshold, count the number of errors and repeat for all possible thresholds). However, the curve resulting from one speaker-specific classifier would be unreliable. A curve that better represents the true detection error trade-off can be obtained by testing a classifier type on a large number of speakers. A more representative curve may be plotted from many speaker-specific classifiers by concatenating the lists of utterance scores. This is applicable only if the scores have the same scale and origin.

### 2.4 Corpora

This section reviews two speaker verification databases: the YOHO Speaker-Verification Corpus and Swiss French PolyVar telephone speech database. The YOHO database was used in this work for developing and
2.4. Corpora

a: An ROC curve illustrates the trade-off between the probability of false acceptances (horizontal axis) against the true acceptance probability or one minus the false rejection probability (vertical axis).

b: The DET curves corresponding to the ROCs shown in (a). The false acceptance probability (horizontal axis) is plotted against the false rejection probability (vertical axis). The conversion from probabilities to the normal deviate scale is shown in (c).

c: The normal deviate is found by computing the percentage area under the normal distribution.

Figure 2.3: Reporting classifier performance on ROC and DET curves
testing the techniques required to apply SVMs to speaker verification. The PolyVar database was reserved as an evaluation database so that when development on YOHO was finished a more conclusive evaluation could be done.

2.4.1 The YOHO voice verification corpus

This database was collected by IIT under a U.S. government contract [Cam95]. It was the first large scale database to be collected under scientifically controlled conditions, with high quality speech data for speaker verification testing at high confidence levels.

The YOHO voice verification corpus is summarised as follows.

- Combination lock phrases, e.g. twenty-six, eighty-one, fifty-seven.
- 138 speakers: 106 male; 32 female.
- Collected using a STU-III electret microphone telephone handset, although the signal was not passed through a telephone channel.
- 96 enrolment utterances per speaker recorded in 4 sessions with 24 phrases per session.
- 40 verification utterances per speaker recorded over 10 sessions at approximately three day intervals with 4 phrases per session.
- Data collected over a period of three months.
- 8kHz sampling rate with 3.8kHz analogue bandwidth.

The speakers spanned a wide range of ages, job descriptions and backgrounds from the New York area, including some non-native-English speakers. Test results on the YOHO database were first reported by Campbell [Cam95] using a variety of different systems.

2.4.2 The PolyVar database

The PolyVar database, together with its big brother PolyPhone, was recorded at IDIAP [CCC+96] over a telephone network. Participants were from all over the French speaking part of Switzerland. Each participant phoned a switchboard several times to record speech from a list of items.

The PolyVar database is summarised as follows.

- The recordings were collected over a telephone so the database inherently contains noisy data.
- Phrases consisted of 5 repetitions of 17 proper nouns and day to day phrases in French, each composed of 3 to 12 phonemes.
- A strictly defined protocol exists. It defines exactly which utterances should be used for training and testing.
- 38 client speakers (24 male and 14 female) each with 85 utterances recorded in 5 sessions with 17 utterances in each session.
- 952 impostor utterances from 56 speakers, each speaker contributing 17 utterances in one session.
- Approximately 1000 test utterances (including both client and unseen impostors) for each client speaker.
Test results on the PolyVar database have been reported by Bengio and Mariéthoz [BM01] using Gaussian mixture models and support vector machines.
Chapter 3

Creating models of speakers

3.1 Modelling strategy

The previous chapter described the way in which a waveform is converted to a sequence of vectors to reduce the data rate of the signal. Once this is done, a model of the speaker must be created. The model is used to obtain an utterance score, which indicates the correspondence between the utterance and the model. The utterance scores are processed using hypothesis testing to obtain the final decision. The full speaker verification process was outlined in chapter 2. This chapter will focus on the speaker models and on obtaining the utterance scores. The different modelling approaches may be divided into two distinct categories: discriminative and generative models.

Discriminative models are optimised to minimise the error on a set of training samples. Classifiers that are discriminative models include multilayer perceptrons (MLPs), polynomial classifiers and support vector machines (SVMs). Discriminative models also include discriminatively trained generative models such as HMMs trained using a maximum mutual information criterion.

Generative models are probability density estimators that attempt to capture all of the underlying fluctuations and variations of the data (in this case the speaker’s voice). Generative models include Gaussian mixture models (GMMs), hidden Markov models (HMMs). We may also include nearest neighbour classifiers in this type of model although strictly speaking they are not likelihood estimators; they do, however, attempt do model the underlying structure of the data by replacing regions containing data with a set of codebook vectors.

3.1.1 Discriminative vs. generative models

To compare the two modelling approaches, consider the sample problem illustrated in figure 3.1. Suppose that the graphs of figure 3.1a are the true probability density distributions of two speakers (these distributions are usually unknown). To build a model that classifies these two speakers using either approach, samples of data must be drawn from each distribution (this corresponds to the feature extraction stage described in section 2.2).

The strategy adopted in generative models is to treat the samples of one speaker independently from the other. For each speaker, a model of the true probability densities is constructed from the sampled data. Figure 3.1b illustrates the probability densities estimated by GMMs. Let the input space be the space occupied by the vectors that are to be classified. To classify some test input vectors, simply choose the model with the greater likelihood given the test input.
3.1. Modelling strategy

Discriminative classifiers are trained to minimise the error on some samples of data. Alternatively, a discriminative classifier models only the boundary between classes and ignores the fluctuations within each class. In Bayes theory, the optimum decision is given by the posterior probabilities that the sampled data are drawn from one of the classes [Bis95, CT91]. If possible, a discriminative classifier should ideally model the posterior distribution. In our example, figure 3.1c shows the posterior distribution of the two classes. In regions of input space where the likelihood that a test vector belongs to one class is significantly greater than the likelihood that it belongs to the other then the posterior probability will be close to one for the first class (and vice-versa). Where the likelihood of both classes are comparable then the posterior probability will depend upon the relative likelihood at a particular point in the input space. For example, in a two class problem, if the likelihoods (and class prior probabilities) that a given point could have come from either class then the posterior probabilities will be exactly 0.5.

The benefit of a discriminative approach is that the class boundaries or the posterior distributions are simpler to model than the within-class likelihood distributions. Where there is no overlap in the probability density functions of the two speakers, i.e. in regions of the input space where it is certain that the sampled feature vector belongs to a particular speaker, the posterior probability is saturated at zero or one. In these regions, a generative model still tries to model the full probability density despite the certainty that the feature vector belongs to one speaker and not the other. Therefore, given a discriminative and a generative model, each with the same number of parameters, the discriminative model has greater capacity to model the variations in the overlap regions of the true probability densities than the "equivalent" generative model. The discriminative model uses its parameters to achieve the highest performance in classification tasks. The generative model is a density estimator of within-class probability densities and does not directly minimise a classification error.

In terms of speaker verification a discriminative model is desirable. Modelling the boundary between client and impostors is a direct and intuitive method of classification, leading to simpler classifiers and/or fewer parameters. Using a method that directly minimises the error rate is particularly appealing. Vapnik [Vap95] divided machine learning into categories of increasing difficulty. These include pattern recognition or classification being the simplest to solve, followed by regression estimation then density estimation as the most difficult of the three. He argued that it is inefficient to solve classification problems indirectly by doing density estimation first.

However, a generative model may be more appropriate. Creating a full model of a client’s voice may make the resulting classifier more robust against unseen impostors. Where a discriminative classifier may have discarded some client information during training, a generative classifier will have retained that information. When classifying data from an unseen impostor the extra information stored by the generative model may yield improved accuracy. Furthermore, training a generative model requires only data samples from the client. Training a discriminative model requires the learning algorithm to look at both the client and impostors in order to construct a posterior distribution model (or decision boundary). This means that the posterior model is tied to a particular set of impostors. If an unseen impostor is not represented in the training data then the posterior distribution may be poor. Furthermore, should the set of impostors be updated at some later date then the discriminative models must be retrained. The generative client models does not require retraining since the likelihood distribution of the impostors can be updated independently.
3.1. Modelling strategy

a: The probability density distribution of two classes is shown. Suppose that these (artificially generated) distributions represent the distribution of acoustic feature vectors of two speakers. To create a model of each speaker data points are first sampled from each distribution (i.e. their speech is recorded and processed).

b: A generative model is created by modelling the data of each class separately. The illustration shows the result of using a three component Gaussian mixture model to learn the data densities at each region of the input space. Classification is achieved by computing the likelihoods of a test data point being generated by each of the distributions and choosing the distribution with the greater likelihood.

c: The posterior probability distribution corresponding to the density distributions in (a) is illustrated. The distribution in this case represents the a posterior probability that a given data point belongs to one of the speakers and not the other. Notice how all of the within-class variations are no longer evident in regions where the data is certain to belong to one of the speakers. This means that only the smaller region where there exists uncertainty requires modelling.

d: A discriminative model directly estimates the posterior distribution. As a result the classifier is often much simpler and/or more compact. Fewer parameters are required in the model since the posterior distribution does not contain as much information as the full probability density functions. It is a discriminant model in that it will only discriminate between these two speakers. The model of this posterior distribution will not give any information on how to discriminate between either of these speakers and a third unseen speaker.

Figure 3.1: The difference between generative and discriminative modelling approaches.
3.2 Generative models

This section reviews generative models that have been used to model speakers. Vector quantisation and GMMs are static models that do not model variations in time. The third model, the HMM, is capable of modelling temporal variations. GMMs and HMMs will be described in detail since they are fundamental to work described later in this thesis. Vector quantisation is not a strictly a generative model, however it is included here since it has links to the GMM.

3.2.1 Vector quantisation

Vector quantisation is a modelling approach that represents clusters in the training data by a set of codebook vectors. There are a variety of methods for finding the codebook vectors. The methods include nearest neighbour and learning vector quantisation [Bis95].

By representing large amounts of data with a small number of vectors the method models the regions of input space that contain data. Although we include vector quantisation under the generative model framework it is not strictly a generative model — non-discriminative would be a better description. It is related to the GMM in that the codebook vectors correspond to the means of the Gaussians. However, unlike GMMs, vector quantisation does not model data densities (one codebook vector may represent an arbitrary number of input vectors). As such, the vector quantisation model does not assign likelihood estimates to test vectors. Instead classification is determined by finding the nearest codebook vector (or vectors) to the test point. Information obtained by classification is therefore restricted to a measure of the distance from the test point to the nearest codebook vector (or nearest k codebook vectors).

In an application to speaker recognition by Soong [Soo90], the codebook vectors were learned by minimising the global distances between the training vectors and their nearest codebook entry. A Euclidean distance measure was used. During classification of an utterance, \( X = \{x_1 \cdots x_N\} \), the distance from the \( i^{th} \) frame, \( x_i \), in the sequence to its nearest codebook vector, \( e_j \), is accumulated to calculate utterance score,

\[
S(X) = \sum_{i=1}^{N} \min_j \|x_i - e_j\|.
\]

(3.1)

In this case, a small utterance score corresponds to a good match to the model while a large score corresponds to a poor match. Note that this is different from all other models described in this thesis.

3.2.2 Gaussian mixture models

The Gaussian mixture model (GMM) is a density estimator and is one of the most commonly used types of classifier. The mathematical form of an \( m \) component Gaussian mixture for \( D \) dimensional input vectors is,

\[
P(\mathbf{x} | M) = \sum_{i=1}^{m} a_i \frac{1}{(2\pi)^{D/2} |\Sigma_i|^{1/2}} \exp \left( -\frac{1}{2} (\mathbf{x} - \mu_i)^T \Sigma_i^{-1} (\mathbf{x} - \mu_i) \right)
\]

(3.2)

where \( P(\mathbf{x} | M) \) is the likelihood of \( \mathbf{x} \) given the mixture model, \( M \). The mixture model consists of a weighted sum over \( m \) unimodal Gaussian densities each parameterised by the mean vectors, \( \mu_i \), and covariance matrices, \( \Sigma_i \). The coefficients, \( a_i \), are the mixture weights, which are constrained to be positive and must sum to one. The parameters of a Gaussian mixture model, \( a_i, \mu_i \) and \( \Sigma_i \) for \( i = 1 \cdots m \) may be estimated using the maximum likelihood criterion via and the iterative Expectation-Maximisation (EM) algorithm [DLR77, RW84]. In general, fewer than ten iterations of the EM algorithm will provide sufficient parameter convergence.
3.2. Generative models

Training GMMs using maximum likelihood leads to a generative model. GMMs are analogous to vector quantisation in that the mean of each Gaussian density can be thought of as a codebook vector. The GMM combines the robust parametric approach of Gaussian density modelling with the arbitrary data modelling approach of the non-parametric vector quantisation model.

A detailed discussion on applying GMMs to speaker modelling can be found in [Rey92]. The basic method is straightforward. GMMs with diagonal covariance matrices are generally used. Although full covariance matrices may be used if desired, the simplification leads to a model with fewer parameters without sacrificing accuracy. Empirical evidence indicates that the accuracy of a full covariance mixture model can be achieved by a diagonal covariance model with a larger number of mixture components.

A GMM is trained using maximum likelihood, to estimate the probability density function, $P(x|M)$, of the client speaker. The probability, $P(X|M)$, that an utterance, $X = \{x_1 \cdots x_N\}$, is generated by the model, $M$, is used as the utterance score. It is estimated by the mean log likelihood over the sequence,

$$S(X) = \log P(X|M) = \frac{1}{N} \sum_{i=1}^{N} \log P(x_i|M),$$

(3.3)

**GMM likelihood ratio**

A better strategy is to use two GMMs instead of one. The first GMM is used to model the client as above. The second GMM is a speaker independent world model, sometimes known as a global background model. The world model is trained on a large number of background speakers and is therefore a general model of speech. The basic idea is that the second GMM should model the properties of speech signals that are common to all speakers. In speaker verification it is desirable to assign high scores to frames, $x_i$, that are specific to the client and low scores to frames that are common to most speakers. This may be achieved by taking the ratio of the client model, $M$, likelihood to the world model, $\Omega$, likelihood on the frame-level. This is equivalent to computing the difference between the client and world model’s mean log likelihood scores for the full sequence,

$$S(X) = \frac{1}{N} \sum_{i=1}^{N} \log \frac{P(x_i|M)}{P(x_i|\Omega)}$$

(3.4)

$$S(X) = \frac{1}{N} \sum_{i=1}^{N} \log P(x_i|M) - \frac{1}{N} \sum_{i=1}^{N} \log P(x_i|\Omega)$$

(3.5)

$$S(X) = \log P(X|M) - \log P(X|\Omega).$$

(3.6)

Consider a frame of data that has a high likelihood of being generated by the client model. If that frame is common in most speech then the world model should have a high likelihood of generating it also. Dividing the client model likelihood by the world model likelihood reduces the score of this frame. As a result, frames that are common to all speakers contribute less the utterance score. On the other hand, if the frame is specific to the client then the world model likelihood will be small. Computing the likelihood ratio will yield a high score and the contribution to the utterance score will be large. In this respect, this method, which we call the GMM likelihood ratio (GMM-LR), may be seen as a simple way of obtaining discrimination from two generative models. The GMM-LR method is better than the GMM method in that the former takes into account the general distribution of speech and scores the utterance according to properties that are specific to the client while avoiding properties that are common in many speakers. It may also be said that the two GMMs are trained from two different data sets, therefore each model contains information that the other does not, thereby increasing the accuracy of the system.
Reynolds [Rey95] used the GMM likelihood ratio approach for speaker verification on the YOHO database. He estimated $P(X|Ω)$ using a world model constructed from a set of speakers selected specifically for each client. The chosen background speakers were divided into two subsets. Selection of speakers in the first subset was made using a log likelihood ratio distance measure to find those statistically nearest to the client. Selecting the subset of speakers that are most similar to the client forces the classifier to focus on the speakers that are more difficult to classify. The second subset consisted of speakers that were maximally spread to cover as many different types of speaker as possible in order to make up the remainder of the world model. The GMM-LR method is a simple yet powerful approach and is often used as a baseline against which other approaches are measured.

### 3.2.3 Hidden Markov models

The third generative model reviewed here is capable of modelling temporal variations in sequences. The hidden Markov model (HMM) formulation by Baum [Bau72] may be described as a finite state generator (figure 3.2). In speaker recognition, each state, $(q_1 \cdots q_n)$, of the HMM may represent phones or other larger units of speech. Temporal information is encoded by moving from state to state along the allowed transitions illustrated. Therefore, the temporal modelling is piecewise stationary. The amount of time spent in each state varies depending upon the data. This allows for variability in speaking rate.

The operation of an HMM is straightforward. Let $M$ denote an HMM such as the one illustrated in figure 3.2a. Consider an utterance, $X = \{x_1 \cdots x_N\}$, to be processed by $M$. Let $q^n_i$ identify the $i^{\text{th}}$ state in the Markov chain occupied at time $n$ (i.e. corresponds to the $n^{\text{th}}$ frame in $X$). At time $n = 0$ the begin state, $q_1$, is occupied. This state is a non-emitting state (that is, no feature vector is associated with it). At time $n = 1$ a transition is made to the first state, $q_1$, and the first feature vector, $x_1$, is emitted with the probability $P(x_1|M,q_1,0)$ (figure 3.2b). With each increment of an $n$ a transition must be taken along one of the edges of the graph with a certain probability (figure 3.2c). At time $n = 2$ a transition must be made to state $q_2$ or back to $q_1$, and the feature vector, $x_2$, emitted. This process continues until $n = N + 1$ at which time the end state, $q_{F}$, another non-emitting state, must be reached. The probabilities of the HMM process are accumulated to obtain the utterance likelihood, $P(X|M)$. In similar fashion to the GMM, the utterance score is $S(X) = \log P(X|M)$.

#### The forward-backward algorithm

In order to compute the utterance likelihood, $P(X|M)$, it is necessary to decompose it into terms that are easily estimated. The theory presented here closely follows the approach by Bourlard and Morgan [BM94].

Let $\{q^1, q^2, \ldots, q^N\}$ denote a valid HMM state sequence. This is one of many possible paths through the Markov chain corresponding to the sequence $X$. The utterance likelihood may be written as a sum over the likelihoods of each unique path, but this is inefficient. A search for all possible paths through the model may be impractical. Instead, examine the occupied states of each unique path at some arbitrary time, $n$. The utterance likelihood may be expressed as a sum over all possible states that can be occupied at time $n$,

$$P(X|M) = \sum_{q^n} P(q^n, X|M),$$

and all possible paths through the HMM remain accounted. By using the basic definitions of conditional probability the term inside the summation can be written,

$$P(q^n, X|M) = P(q^n, X^n|M)P(X^{n+1}|q^n, x_{1}^{n}, M) = \alpha(l)\beta_{n+1}(l)$$

(3.8)
3.2. Generative models

a: A simple four state HMM. Each state, $q_i$, models a different section of the speech signal. Transitions are made from one state to the next along the indicated edges. Self-transitions that begin and end at the same state allow that state to be occupied for a variable amount of time.

b: The state dependence diagram of a first-order HMM. A first order HMM is one in which the state occupied at time $n$ is dependent upon the previously occupied state only. Likewise the symbol, $x_n$, emitted at time $n$ only depends upon the state, $q_n$, occupied at that time.

c: The probabilities associated with the HMM. $\epsilon$ is the probability of exiting one state to a different state. Subsequently $(1-\epsilon)$ is the probability of staying in the same state. Upon entering a state the symbol $x_n$ is emitted with probability $P(x_n|q_n)$. This probability is typically estimated by a Gaussian mixture model. The begin and end states are non-emitting states that do not have an emission probability associated with them.

Figure 3.2: Finite state generator HMM
where $X_i^n$ represents the set of feature vectors between times $c$ and $d$. Consider the term known as the forward probability,

$$\alpha_n(l) = P(q_l^n, X_l^n | M) \quad (3.9)$$

This is the probability that state $q_l$ is occupied at time $n$ having generated the feature vectors up to that time. A simple recursion, known as the forward recursion, to calculate $\alpha_n(l)$ is,

$$\alpha_n(l) = \left[ \sum_k \alpha_{n-1}(k) P(q_l | q_k, M) \right] P(x_n | q_l, M) \quad (3.10)$$

The starting condition is $\alpha_0(l) = 1$ in which the starting state at time $n = 0$ is a non-emit state, $q_l$.

To understand how this recursion arises, consider all the paths that pass through state $l$ at time $n$, $q_l^n$ (see figure 3.3). The value of $\alpha_n(l)$ is an accumulation of the likelihoods of those paths arriving at state $l$ at time $n$. Therefore, the likelihoods of those paths must have been multiplied by the emission probability $P(x_n | q_l)$. They must also have been multiplied by the transition probability of getting to this state, $P(q_l | q_k)$, from the previous state, $k$. The value of $\alpha_{n-1}$ is itself an accumulation of the likelihoods of paths reaching $k$ at time $(n-1)$. To accumulate the likelihoods of paths arriving at $l$ at time $n$ simply sum over the $\alpha$’s of all possible previous states multiplied by the correct transition probability then multiply by the state emission probability.

Written fully, the forward recursion is in fact,

$$\alpha_n(l) = P(q_l^n, X_l^n | M) \quad (3.11)$$

$$= \sum_k P(q_k^{n-1}, X_1^{n-1} | M) P(x_n | q_l^n, q_k^{n-1}, X_1^{n-1}, M) P(q_l^n | q_k^{n-1}, X_1^{n-1}, M) \quad (3.12)$$

where $P(q_k^{n-1}, X_1^{n-1} | M) = \alpha_{n-1}(k)$. Equation (3.12) becomes (3.10) by applying the following simplifying assumptions.

1. First order Markov process:

$$P(q_l^n | q_k^{n-1}, X_1^{n-1}, M) = P(q_l^n | q_k^{n-1}, M) \quad (3.13)$$

The state at time $n$ is dependent upon the state at time $(n-1)$ only and not on the feature vectors or on any states that came before time $(n-1)$. This term can be estimated by counting the number of state transitions in the training data and is known as the transition probability.

2. Conditional independence of observations given the state:

$$P(x_n | q_l^n, q_k^{n-1}, X_1^{n-1}, M) = P(x_n | q_l^n, q_k^{n-1}, M) \quad (3.14)$$

The feature vectors emitted at time $n$ are not related to the vectors that came before it. The feature vectors are only dependent upon the current and previous one state. They are emitted on the transition from $q^{n-1}$ to $q^n$. However,

3. emission-on-transition probabilities are reduced to emission probabilities,

$$P(x_n | q_l^n, q_k^{n-1}, M) = P(x_n | q_l^n) \quad (3.15)$$

The feature vectors are assumed to be dependent only on a single state and independent of the model so that the term becomes the probability that a state will emit some acoustic vector. This has the effect of reducing the number of parameters in the models and converts a transition based modelling technique to a state based technique.
The three possible paths that can be taken to reach state $q_i^*$ are shown by the coloured arrows. The forward recursion for this state accumulates the likelihoods of the three paths and can be calculated quickly as a sum over the values of the forward recursion of the previous two states, $q_i'$ and $q_i''$, weighted by the transition probabilities indicated and multiplied by the current emission probability, $P(x_i|q_i)$. 

Figure 3.3: Illustration of the forward recursion alignment matrix.
4. Time independent transition probabilities,

\[ P(q_{t}^{n} | q_{t-1}^{n-1}) = P(q_{t} | q_{t-1}) \]  

(3.16)

The probability of making a transition from one state to the next is assumed to be dependent upon the originating and destination states and independent of when the transition occurs within the utterance.

Figure 3.2b illustrates the conditional dependencies within generator HMMs that arise from these assumptions. It is worth noting that these assumptions are here for computational convenience. By the nature of language, there exist strict dependencies throughout an entire speech signal. The assumptions of the HMM relaxes these dependencies to a high degree.

Any assumption that removes the dependence of one segment of acoustics from another within the same sentence has limited validity. The assumptions listed take the extreme case where adjacent feature vectors are assumed conditionally independent.

Using the same methods and assumptions the backward recursion for \( \beta_{n}(l) \) can be obtained.

\[ \beta_{n}(l) = \sum_{k} P(q_{k}^{n} | q_{l}, M)P(x_{n+1} | q_{k}, M)\beta_{n+1}(k) \]  

(3.17)

The initialisation is \( \beta_{N}(l) = 1 \) if the state, \( q_{l} \), connects directly to a non-emitting HMM final state, \( q_{F} \), else \( \beta_{N}(l) = 0 \) if it is not a valid end state.

An estimate for the utterance likelihood, \( P(X | M) \), is obtained by combining equations (3.7), (3.8), (3.10) and (3.17). This decomposition is known as the forward-backward algorithm and enables HMMs to be used efficiently. In some systems the Viterbi approximation is used. This involves replacing the summations in the recursions with a max function.

**Training HMMs by Maximum Likelihood Estimation**

Maximum likelihood is a popular criterion for estimating the parameters of a generative HMM. As its name suggests, the goal is to estimate the parameters, \( \theta \), of the model by maximising the utterance likelihood, \( P(X | M, \theta) \), that the model, \( M \), can generate the sequence, \( X \).

When training an HMM on speech, the model topology, is known from the transcription. However, it is not known which frames in the utterance, \( X \), correspond to which states in the HMM. That is, the precise alignment between the feature vectors and the HMM states (called the state trajectory) is unknown. The training task is, therefore, to estimate the parameters of the model such that the utterance likelihood is maximised while allowing the HMM to determine its own state trajectory. This can be achieved using the following iterative scheme:

1. Guess the initial parameters, \( \theta \).

2. Use \( \theta \) to obtain a set of likelihoods at each time frame within each utterance.

3. Use the forward-backward algorithm with the likelihoods in the previous step to evaluate new utterance likelihoods defined by equation (3.8), \( P(q_{t}^{n}, X | M, \theta) \).

4. Using the new likelihoods, evaluate a new set of parameters \( \theta' \). For the case where states are modelled by a single Gaussian the new parameters are simply the new means and variances [BM94].

5. Set \( \theta \) to be \( \theta' \).

6. Repeat from step 2 until convergence to a locally optimal solution.
The algorithm is an example of a more general algorithm called the Expectation-Maximization (EM) algorithm and is guaranteed to converge [DLR77].

3.3 Discriminative models

The models reviewed in this section are related to linear discriminative classifiers — the idea of putting a straight line through a set of points to separate them into two classes is the most basic discriminative model. The relation to a linear discriminant is obvious in polynomial classifiers and SVMs. In neural networks each node is effectively a linear classifier. Under the correct conditions, each of these classifiers are capable of estimating the posterior distribution.

3.3.1 Support vector machines

The support vector machine (SVM) is a discriminative classifier that is simple in concept but has some extensions that make it very powerful. Underlying every SVM is a linear classifier but through the use of kernel functions the SVM can make non-linear decisions. Choosing an appropriate kernel function allows the SVM to mimic other classifiers that are based on linear discriminants. For example, using a polynomial kernel the SVM output function is similar to that of the polynomial classifier. The kernel function can even allow the SVM to classify variable length sequences. Chapter 4 describes the SVM in detail.

3.3.2 Polynomial classifiers

The polynomial classifier [Spe67, CA99] is a discriminative model that uses polynomial functions to estimate the decision boundaries between classes. It is a binary classifier where the output can be interpreted as a posterior probability. Its underlying mathematical formulation makes it very similar to an SVM with a polynomial kernel.

In this method the input vectors, x, which are the vectors presented to the classifier and exist in the input space, are mapped onto a manifold embedded in a high dimensional space called the feature space, \( \Phi, x \rightarrow \Phi(x) \). For example, a second order polynomial would map the input vector x onto the feature vector

\[
\Phi(x) = \begin{bmatrix}
1 \\
x_1 \\
x_2 \\
\vdots \\
x_1x_2 \\
\vdots \\
x_1^2 \\
x_2^2 \\
\vdots
\end{bmatrix}
\]  

(3.18)

where \( x_i \) is the \( i \)th coefficient of x and the components of \( \Phi(x) \) are the coefficients of a quadratic expansion. A linear classifier is constructed that best separates the feature vectors of the client, \( \Phi(x_{\text{client}}) \), from those of the background speakers, \( \Phi(x_{\text{world}}) \) in the high dimensional feature space. Let \( x_i^j \) denote the \( i \)th frame of data corresponding to speaker \( j \). Let the desired output of the classifier be one for the client vectors and
zero for background speakers. Using a squared-error cost function to penalise the difference of the actual classifier output from the desired output, the empirical risk is,

$$R_{emp} = \sum_{j \in \{\text{client}\}} (f(\Phi(x_j^i)) - 1)^2 + \sum_{j \in \{\text{world}\}} f(\Phi(x_j^i))^2.$$  

(3.19)

The aim is to find a form of the function $f$ that minimises the empirical risk. A general linear boundary in the feature space can be expressed in the form, $f(\Phi(x_i)) = \Phi(x_i)w$, where $w$ are the adjustable parameters that define a hyperplane in the polynomial feature space. The optimum decision boundary is found when the empirical risk is minimised.

$$w_0 = \arg\min_w \sum_{j \in \{\text{client}\}} (\Phi(x_j^i)w - 1)^2 + \sum_{j \in \{\text{world}\}} (\Phi(x_j^i)w)^2$$  

(3.20)

and may be computed as follows. Define $M_{\text{client}}$ to be a matrix whose rows are the polynomial expansion of the client’s input vectors.

$$M_{\text{client}} = \begin{bmatrix} 
\Phi(x_1^{\text{client}})^T \\
\Phi(x_2^{\text{client}})^T \\
\vdots \\
\Phi(x_n^{\text{client}})^T 
\end{bmatrix}$$  

(3.21)

Likewise the matrix for background speakers is

$$M_{\text{world}} = \begin{bmatrix} 
\Phi(x_1^{\text{world}})^T \\
\Phi(x_2^{\text{world}})^T \\
\vdots \\
\Phi(x_n^{\text{world}})^T 
\end{bmatrix}$$  

(3.22)

Define the matrix $M$ as the concatenation of $M_{\text{client}}$ and $M_{\text{world}}$

$$M = \begin{bmatrix} 
M_{\text{client}} \\
M_{\text{world}} 
\end{bmatrix}$$  

(3.23)

Equation (3.20) becomes,

$$w_0 = \arg\min_w \|Mw - o\|_2$$  

(3.24)

where $o$ is a vector of ones and zeros (the desired outputs). Applying the method of normal equation [GvL89] we get,

$$M^TMw = M^To$$  

(3.25)

which becomes,

$$(M^T_{\text{client}}M_{\text{client}} + M^T_{\text{world}}M_{\text{world}})w = M^T_{\text{client}}1$$  

(3.26)

where $1$ is a vector of ones. This is the basis of the training method. To solve for $w$, defining,

$$R = M^T_{\text{client}}M_{\text{client}} + M^T_{\text{world}}M_{\text{world}}$$  

(3.27)

compute the Cholesky decomposition of $R$, $R = L^TL$ and solve $L^TLw = M^T1$ by using a triangular solver twice.

Applying polynomial classifiers to speaker verification is similar to the application of the GMM. The learned function $f(\Phi(x))$ assigns a score to each vector in the utterance $X = \{x_1, \cdots, x_N\}$. The utterance score is mean of the individual frame scores,

$$S(X) = \frac{1}{N} \sum_{i=1}^{N} f(\Phi(x_i))$$  

(3.28)

where $x_i$ is the $i^{th}$ frame in the utterance.
3.3. Discriminative models

3.3.3 Neural networks

An artificial neural network [Bis95] is a powerful tool for regression and classification. There are many types of neural network but we shall focus on two specific types that have been applied to speech. The multilayer perceptron (MLP) is a feed-forward network that incorporates little or no temporal information. The recurrent network, extends the MLP by incorporating feedback loops in its architecture to incorporate a greater amount of temporal information.

Multilayer perceptors

The architecture of an MLP is illustrated in Figure 3.4. Each node computes a linear weighted sum over its input connections, where the weights of the summation are the adjustable parameters. A transfer function is applied to the result to compute the output of that node. Commonly used transfer functions include linear, tanh, sigmoid and softmax functions. The weights of the network are estimated by gradient descent in a process called back-propagation. It is well known that an MLP with a non-linear transfer function and sufficiently large number of nodes in the hidden layer may approximate any functional mapping from input to output. This is why MLPs are considered a useful tool. Furthermore, with appropriate constraints, an MLP may be used to estimate posterior probabilities directly [Bri89, Bis95, Gis90, BW88].

![Diagram of a multilayer perceptron](image)

**Figure 3.4:** A multilayer perceptron
3.4. Making generative models discriminative

An MLP for speaker verification would have only one output node since the task is only to score the frames of an utterance. In the same fashion as the polynomial classifier and GMM, the utterance score is the mean classifier output over the complete utterance [OM90].

Temporal neural networks

Researchers have attempted to incorporate temporal processing by building networks with more complex architectures. For example, time delay neural networks (TDNN) [LWH90] incorporate a short term memory of the most recent set of input vectors. A set of buffers inserted between layers of the network store the \( n \) most recent set of activations of the previous layer so that the stored activations can all be presented to the next layer simultaneously. The weights of the connections to the next layer are tied in such a way that the network interprets each stored vector of activations in the same way.

Another method introduces feedback loops into the network architecture by connecting output units, via a time delay, back to the input units. In this way the network is allowed to construct its own internal representation of an utterance. These types of network are called recurrent networks and have been used extensively in speech recognition tasks [Rob94, RHR96, HCR+95, Ker97]. However, recurrent networks are only suitable for implementing short-term memories. It has been proven that training recurrent networks becomes increasingly difficult as the length of the sequences and the duration of temporal dependencies increases [BSF94]. This is primarily an effect of error propagation in gradient descent methods.

3.4 Making generative models discriminative

Generative models have properties that discriminative models do not and vice-versa. A combination of properties from each may be desirable for a given task. For example, generative models can handle arbitrary length sequence data efficiently through the use of HMMs while discriminative models described above are somewhat lacking in this respect. Conversely, discriminative models make better classification models — they are explicitly trained to suppress incorrect classifications. As a result, much work has gone into combining the two modelling strategies in order to build a model with properties of both.

There are many different ways of building generative models into a discriminative framework. These methods range from combining a generative model with a discriminative classifier to adjusting the parameters of a generative model using a discriminative objective function. This section will examine briefly a few of the different ways of making generative models discriminative.

3.4.1 Radial basis function networks

Our first example of a classifier that combines a generative model in a discriminative framework is the radial basis function (RBF) network [Bis95]. It combines the generative GMMs with the discrimination of an MLP. The mathematical form of an RBF network is almost identical to that of several GMMs. The RBF has a two layer topology similar to that of a two layer neural network. The difference is that the nodes of the hidden layer each consist of a unimodal Gaussian. The output layer is the same as the output layer of a neural network and computes weighted summations of the Gaussians. An RBF network with an arbitrary number of outputs is illustrated in figure 3.5. Note that in an application to speaker verification, only one output unit is required to score an utterance.
3.4. Making generative models discriminative

Given an input vector $\mathbf{x}$, the $j$th output, $f_j(\mathbf{x})$, of a network with $m$ hidden units is,

$$f_j(\mathbf{x}) = \sum_{i=1}^{m} a_i^j \frac{1}{(2\pi)^{D/2} |\Sigma_j|^{1/2}} \exp \left( -\frac{1}{2} (\mathbf{x} - \mu_j)^T \Sigma_j^{-1} (\mathbf{x} - \mu_j) \right).$$  \hfill (3.29)

Herein lies the similarity to Gaussian mixture models. Equation (3.29) for the output at one node of the RBF is identical to the output of a Gaussian mixture model, equation (3.2), less the constraints on the coefficients $a$. An RBF network has an arbitrary number of outputs all of which compute their outputs from a weighted sum over the same set of Gaussian densities. This means that an RBF network with $N$ outputs is equivalent to $N$ Gaussian mixture models with their means and covariances tied together. An RBF network with one output differs from a GMM only by the way the Gaussian component mixture weights are learned. The GMM uses those parameters to model the data while the RBF network uses them to minimise the number of errors on the training data. Note that the relationship with vector quantisers and GMMs also apply to RBF networks in the sense that the hidden layer of the RBF network consists of a set of codebook vectors with corresponding Gaussian densities.

The RBF network is also a simple example of stacking. Stacking is where the output of one classifier is used as the input to a second classifier. In the case of the RBF network a single layer neural network post-processes the output of a set of Gaussian densities. Stacking can sometimes attain lower error rates than the two classifiers by themselves.

An RBF network can be trained in one of two ways. It can be trained entirely by minimising the empirical risk using gradient descent in which case the RBF is a completely discriminative model. Alternatively, it can be trained by a combination of gradient descent, for learning the weights of the output layer, and maximum likelihood using unsupervised expectation-maximisation (EM) algorithm, for learning the means and covariances of the Gaussians. For the latter, the Gaussians retain their generative model nature, estimating the data densities, while the output weights provide the discrimination.

### 3.4.2 GMM-LR/SVM combination

The combination of a GMM likelihood ratio with an SVM to build a generative model into a discriminant framework was done by Bengio and Mariéthoz [BM01]. The method is similar to the RBF network. The output score of a GMM likelihood ratio, from equation (3.6), is

$$S(\mathbf{x}) = \log P(\mathbf{x}|\mathbf{M}) - \log P(\mathbf{x}|\mathbf{O}).$$  \hfill (3.30)
By Bayes Decision rule, this is optimal so long as the probabilities are estimated perfectly. Bengio proposed that the probability estimates are not perfect and a better version would be,

$$S(X) = a \log P(X|M) - b \log P(X|\Omega) + c$$  \hspace{1cm} (3.31)

where $a$, $b$ and $c$ are adjustable parameters. Given a set of training data and labels, these parameters may be estimated using any learning algorithm. Bengio used an SVM to learn these parameters. The input to the SVM is the two-dimensional vector made up of the log likelihoods of the client and world models. The similarity of this classifier to the RBF network can be seen in figure 3.6. The two hidden units correspond to the outputs of the client and world GMMs.

### 3.4.3 HMM/MLP hybrids

The HMM/MLP hybrids combine the efficient temporal processing of HMMs with the discriminative power of an MLP. In the approach adopted by Bourlard and Morgan [BM94] the discriminative HMM estimated the utterance posterior probabilities, $P(M|X)$, instead of the likelihoods. In a fashion not too dissimilar to the generative HMMs, the utterance posterior probability may be decomposed into frame-level posterior probabilities that may be estimated by MLPs. However, instead of deriving an exact forward-backward algorithm they used a Viterbi decoding to estimate the most likely state trajectory and derive a set of targets to train the MLP.

Bourlard et al. [BKM96] later derived the corresponding forward and backward recursions for the hybrid classifier. Like the recursions in the generative HMM they allow the frame posterior probabilities to be estimated and improved from an initial guess. This leads to an iterative training procedure that is identical to the one outlined in section 3.2.3.

Recurrent networks have also been used to attain discrimination in HMMs [RF90, Rob94, RHR96, HCR+95, Ker97]. Bengio [BF96] proposed a new discriminative HMM with an architecture similar to that of a recurrent neural network, called Input/Output HMMs, that could also be trained using an EM algorithm.

In summary, the hybrid HMM/neural network methods replace the likelihood estimating GMMs in generative HMMs with posterior probability estimating neural networks. With appropriate modifications in the mathematics, the utterance posterior probability is obtained instead of the likelihood.
3.4.4 Discriminative training of HMMs

The techniques reviewed so far incorporate discrimination into generative models by directly combining a generative model with a discriminative classifier. A generative model may be made discriminative by choosing a different optimisation criterion. This is the case for HMMs where the maximum likelihood criterion may be replaced with a maximum mutual information criterion. Mutual information [CT91] is a concept from information theory. It is shown below that the maximum mutual information (MMI) criterion can be considered as an extension of the maximum likelihood criterion. It enables the parameters of the HMM to be adjusted in a discriminative way so that more emphasis is placed on modelling the boundaries between classes than on modelling within class variations.

**Mutual information**

To introduce mutual information we first define a measure of information called entropy. The entropy of a random variable \(X\),

\[
H(X) = -\sum_x P(x) \log P(x)
\]

is a measure of the uncertainty of the random variable and can be used as a measure of the amount of information that must be stored in order to describe the outcome \(x\) of \(X\).

In addition, we must define the conditional entropy,

\[
H(X|Y) = -\sum_{x,y} P(x,y) \log P(x|y)
\]

which is the uncertainty in \(X\) given that we know the outcome of another random variable \(Y\). It is a measure of the amount of information that must be stored to describe the outcome of \(X\) given that we know the outcome of \(Y\).

The mutual information can now be defined as the amount of information that one random variable possesses about another and can be specified as the change in the amount of information that must be stored before and after observing \(Y\).

\[
I(X;Y) = H(X) - H(X|Y) = \sum_{x,y} P(x,y) \log \frac{P(x|y)}{P(x)}
\]

(3.34)

(3.35)

**Achieving discrimination in HMMs using MMI**

In section 3.2.3, the HMM training objective was to maximise the likelihood that the client model, \(M\), could have generated the utterance \(X\). When training HMMs using MMI, the mutual information, \(I(X;M_i)\), between the utterance, \(X\), and the client model, \(M_i = M_{\text{client}}\), is maximized. Unlike maximum likelihood, parameter estimation by MMI can only be optimised by gradient descent. By computing the gradients of \(I\) with respect to the parameters, we can see how MMI and maximum likelihood criteria are related and the way in which discrimination arises [Bro87]. From equation (3.35) the mutual information between the utterance and the client model is,

\[
I(X;M_i) = \sum_{M_i, X} P(M_i, X) \left( \log P(X|M_i) - \log P(X) \right)
\]
Since $P(M_i,X)$ is unknown we choose to maximize the terms in brackets making the assumption that the sample is representative. Including the model parameters explicitly, the terms to be differentiated are

$$f(X,M_k) = \log P(X|M_k;\theta) - \log \sum_j P(X|M_j;\theta)P(M_j).$$

In a speaker verification $j = \{\text{client,world}\}$. Consider the derivative with respect to $\theta$, the parameters of the client model.

$$\frac{df(X,M_k)}{d\theta} = \frac{1}{P(X|M_k;\theta)} \frac{d}{d\theta} P(X|M_k;\theta) - \frac{\sum_j \frac{d}{d\theta} P(X|M_j;\theta)P(M_j)}{\sum_j P(X|M_j;\theta)P(M_j)}$$

$$= \frac{1}{P(X|M_k;\theta)} \frac{dR_k}{d\theta} - \frac{1}{\sum_j P(X|M_j;\theta)P(M_j)} \sum_j \frac{dR_j}{d\theta} P(M_j)$$

(3.38)

where $R_j = P(X|M_j;\theta)$. Dropping the terms in the summation that are constant with respect to the parameters of the client model,

$$\frac{df(X,M_k)}{d\theta} = \left(\frac{1}{P(X|M_k;\theta)} - \frac{P(M_k)}{\sum_j P(X|M_j;\theta)P(M_j)}\right) \frac{dR_k}{d\theta}$$

(3.39)

The result obtained by performing the same calculation for maximum likelihood estimation is

$$f(X,M_k) = \log P(X|M_k;\theta)$$

$$\frac{df(X,M_k)}{d\theta} = \frac{1}{P(X|M_k;\theta)} \frac{dR_k}{d\theta}$$

(3.40)

Compare the result of the calculation using MMI with the result that is obtained using maximum likelihood. The first term of the derivative for MMI is identical to the maximum likelihood derivative. The second term contains a summation over all of the models including the incorrect ones. By subtracting this term, the gradient is smaller when the data is confusable, i.e. when the likelihoods of correct and incorrect models are comparable. This makes the parameters more finely tuned on confusable data. Hence, MMI objective function forces the parameters to focus on the class boundaries. The principle is similar to that of the GMM-LR technique where the log likelihood of the world model is subtracted from the log likelihood of the client model to make the class separation larger when the data is not confusable than when the data is confusable.

**Frame discrimination**

The class of frame discrimination objective functions by Kapadia [Kap98] is an extension of the MMI technique. In short, the mutual information objective function may be manipulated to introduce a greater number of valid confusable states at each frame in the HMM. This leads to a discriminative model that is better trained on the confusable data.

Kapadia also showed that this class of objective functions is similar to the frame discrimination objective functions used by the HMM/MLP hybrids described in 3.4.3. The objective function adjusts the HMM state probability density estimators to approximate the posterior distribution instead of the likelihood distribution.

### 3.5 Summary

This chapter reviewed many of the classification methods used in speech with some emphasis on the application to speaker verification. Different types of discriminative and generative models and the different
ways they may be combined were examined. An illustration showing the relationships between the methods discussed in the chapter is shown in figure 3.7. Note that the illustration does not attempt to show all possible connections. For example, there should be a link from polynomial classifiers and HMMs to a method called polynomial networks [CB00]. There is another crucial missing link: the link between SVMs and generative models. The precise nature of this link will be discussed in section 4.5.

3.5.1 Relative accuracy

To conclude this review we shall try to give an indication of the relative accuracy of the modelling strategies described in this chapter. Despite their successful application in speech recognition, MLPs give the worst performance of the techniques described in chapter 3. Mak et al. [MAS94, Mak95] reported that RBF networks are more robust than recurrent networks, which in turn are more robust than MLPs. Yu [Yu95] and Farrall [Far94] reported that vector quantisers and RBF networks both give better accuracy than MLPs, while RBF networks [OM91] and GMMs [Far94] both out perform vector quantisation methods. The GMM [Rey92] is one of the most accurate classifiers for speaker recognition. In text dependent recognition tasks, HMMs [NML98, Tis91] are more accurate than GMMs since the former can better model temporal variations.

The first application of SVMs to speaker identification was by Schmidt and Gish [SG94]. Their experiments on the Switchboard database yielded results that improved upon a modified BBN GMM system [GS94]. More recently, the combined GMM/SVM system [BM01], evaluated on PolyVar and reviewed in section 3.4.2, was shown to improve the accuracy of the underlying GMM. The polynomial classifier, reviewed in section 3.3.2, has been combined with SVMs in the form of a sequence kernel [Cam02] that yields highly competitive results with respect to the 1998 NIST evaluations [PM98, DPMR00].

This thesis will focus on applying SVMs to text independent speaker verification using YOHO for development work and PolyVar for the final evaluation. The polynomial classifier and the GMM will serve as the baseline classifiers. For the final evaluation, the GMM/SVM will be the state of the art classifier since it currently yields the best performance on PolyVar.
3.5. Summary

Discriminant methods
- Linear discriminant
- Perceptron
- Polynomial classifier
- Support vector machines
- Multilayer perceptron
- Recurrent neural networks
- Maximum margin polynomial expansion
- Polynomial kernel
- Tanh kernel

Combined methods
- Bengio’s GMM/SVM
- Radial basis function networks
- HMM/neural net hybrids
- Discriminatively trained HMMs
- Input/output HMM
- Discriminatively trained models

Generative methods
- Vector quantisation
- Gaussian models
- Hidden Markov models

Figure 3.7: The relationships between discriminative and generative models

*Strictly, vector quantisation is not a generative model. A better description is non-discriminative.
Chapter 4

Support vector machines

4.1 Introduction

The aim of this chapter is to provide an understanding of the fundamental ideas behind support vector classification [Vap95]. We explain the concept of maximum margin and how it leads to a regularised classifier that can be used to classify high dimensional data. In addition to the basics underlying the SVM, we explain how it is capable of automatically determining the complexity of its solutions according to the data by discarding easily classified samples and keeping only the most difficult to classify samples. In speaker verification this concept is similar to idea that is used in GMM-LR methods to introduce discrimination into generative GMMs (see section 3.2.2) by selecting a subset of background speakers that are most similar to the client. We also cover the necessary background on how a linear classification algorithm is extended to non-linear decisions via kernel functions and how the idea can be extended to classify variable length sequences. The variable length sequence kernels we review in this chapter is the general set of score-space kernels [SGN01] of which we focus on the likelihood score-space or Fisher [JH99] kernel and the likelihood ratio score space kernel. Other properties of SVMs are also examined in a comparison with RBF networks. We begin by introducing the concept of risk minimisation before moving on to the SVM.

4.2 Machine learning by risk minimisation

Suppose we have a set of pair-wise mappings (training data) \( \mathbf{x}_i : \rightarrow y_i \) where \( \mathbf{x}_i \) is an input vector such as those obtained from LPC analysis (see chapter 2) and \( y_i \) is the desired classifier output or class label that is associated with \( \mathbf{x}_i \). We want a machine to use the information in these pair-wise mappings to automatically assign labels to new input vectors.

The simplest method to achieve this goal is to have the machine remember all of the training data. To classify a new input vector, \( \mathbf{x} \), we would look in the training data for the closest vector and give it the same label. This may be all you can do if there are only a few training samples but it is cumbersome when the quantity of training data is large. Instead the machine can perform the mapping \( \mathbf{x} : \rightarrow f(\mathbf{x}, \mathbf{w}) \) where \( f \) is the output of the classifier (usually some approximating function) and \( \mathbf{w} \) is a vector of parameters that can be adjusted to make the classifier fit the data. Crudely speaking, the machine has learned the training data when \( y_i = f(\mathbf{x}_i, \mathbf{w}) \) for as many \( i \)'s as possible. Formally, we want to minimise the number and magnitude of errors, \( y_i - f(\mathbf{x}_i, \mathbf{w}) \), between the outputs of the classifier and the desired targets over the training set. We can vary the severity of errors of different magnitudes by using some cost function, \( L \), and minimise
4.3. Binary linear decision boundaries

Figure 4.1: Graph of risk vs. complexity.

the classifier’s expected error, which is approximated by the empirical risk [VC74],

\[ R_{\text{emp}}(w) = \frac{1}{N} \sum_{i} L(y_i - f(x_i, w)) \]  \hspace{1cm} (4.1)

Generally the distribution of the training data is unknown. This makes choosing the number of parameters, \( w \), a challenge. When there are too few parameters the classifier cannot reach its peak performance. When there are too many parameters then direct minimisation of the empirical risk may lead to over-training and poor generalisation. Over-training occurs when a classifier has too many degrees of freedom and learns all of the nuances of the training data including irrelevant information resulting from noise. To prevent over-training we need to introduce regularisation. It is possible to minimise the empirical risk with some complexity term, \( T(w) \), which increases as the empirical risk decreases. Minimisation of the regularised risk,

\[ R_{\text{reg}}(w) = T(w) + cR_{\text{emp}}(w) \]  \hspace{1cm} (4.2)

(where \( c \) is a constant) strikes a balance between performance on the training vectors and classifier complexity. The balance is illustrated in figure 4.1. The empirical risk decreases to zero with sufficiently high classifier complexity but the complexity term prevents the classifier from reaching the stage where it is over-trained.

Equation (4.1) is used to define the objective function in many discriminative classifiers such as MLPs and polynomial classifiers. The SVM minimises (4.2) instead. Let us examine the linear SVM classifier in detail.

4.3 Binary linear decision boundaries

This review of SVMs closely follows the tutorial on SVMs by Burges [Bur98]. The SVM is formulated in such a way that it is only capable of discriminating between two classes. Most classification tasks typically involve more than two classes. However, we can reduce any problem to a set of two-class (binary) problems with a divide and conquer strategy (see section 4.7.3). We shall restrict our discussion to solving binary problems.

Let the two classes of the binary problem be labelled \( +1 \) and \( -1 \). In terms of classification the magnitude of the classifier’s output is unimportant (for now) so it suffices to say that an input vector shall be
placed into the +1 class when \( f(x, w) > 0 \) and -1 class when \( f(x, w) < 0 \). The function, \( f(x, w) \), therefore defines a decision boundary that splits the two classes.

Choosing the best function, \( f(\cdot) \), depends on the training data but we will not be able to proceed without making some prior assumption about the relationship between the parameters and the input vectors. For the sake of simplicity we choose a linear relation,

\[
f(x, w) = x \cdot w + b, \tag{4.3}
\]

where \( b \) is a constant. Consider the example shown in figure 4.2a, which uses two dimensional input vectors. Since we have chosen a linear relation the decision boundary must be a straight line. In the example illustrated the decision boundary classifies all the training samples correctly and hence the error is zero. However, there are an infinite number of decision boundaries all of which have zero error (figure 4.2b). An intuitive choice for the best decision boundary is the line that is “exactly half way between the two classes”.

### 4.3.1 Maximising the margin

To find the “middle-line” consider two parallel lines both of which separate the two classes without error. Keeping them parallel but allowing them to rotate, move them as far apart from each other as possible without either line making an error. The result is figure 4.2c. These two lines define the margin. By moving them apart the margin, the distance between the lines, is maximised. The chosen decision boundary is the line that splits the margin in half.

Alternatively, the maximum margin can be found by constructing a convex hull around each class (figure 4.2e). The normal to the decision boundary and margin edges are parallel to the direction of the shortest line between the two convex hulls. With the margin edges touching the convex hulls we arrive at the same decision boundary.

Let us fix the output of the classifier at the edges of the margin to be +1 and -1 (figure 4.2d). The perpendicular distance between the edges of the margin can be found by computing the dot product between the unit vector perpendicular to the decision boundary, \( \frac{w}{||w||} \), and any vector from one edge of the margin to the other, \( (x_2 - x_1) \). By substituting \( x_1 \cdot w + b = -1 \) and \( x_2 \cdot w + b = 1 \), the equations for any point on the two margin edges, we can calculate the margin width to be

\[
\frac{w \cdot (x_2 - x_1)}{||w||} = \frac{2}{||w||}. \tag{4.4}
\]

From equation (4.4) we can see that the decision boundary, found by maximising the margin, may be equivalently found by minimising \( ||w||^2 \) subject to the inequalities

\[
(x_i \cdot w + b) y_i \geq 1 \tag{4.5}
\]

for all \( i \). The optimisation problem can be solved by introducing a Lagrange multiplier for each constraint, \( \alpha_i \), and a Lagrangian,

\[
W_{\text{primal}} = \frac{1}{2} ||w||^2 - \sum_i \alpha_i (y_i (x_i \cdot w + b) - 1). \tag{4.6}
\]

Differentiation with respect to \( w \) and \( b \) leads to

\[
\sum_i \alpha_i y_i = 0 \tag{4.7}
\]

and

\[
w = \sum \alpha_i y_i x_i. \tag{4.8}
\]
4.3. Binary linear decision boundaries

Figure 4.2: How a support vector machine works on separable data. (Source [Bur98].)
4.3. Binary linear decision boundaries

Substituting (4.7) and (4.8) into (4.6) eliminates \( w \) and \( b \) to arrive at the dual optimisation problem to maximise

\[
W_{\text{dual}} = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i \cdot x_j
\]  

subject to \( \alpha_i \geq 0 \) for all \( i \) and \( \sum_i \alpha_i y_i = 0 \). The objective function is maximised by adjusting the Lagrange multipliers, \( \alpha \). It is a quadratic programming problem that can be solved using standard techniques [Lue84]. The decision boundary may be written as follows,

\[
f(x) = \sum_i \alpha_i y_i (x \cdot x) + b
\]  

4.3.2 Support vectors — memorisng the training data

We see that the solution for the decision boundary is some linear combination of the input vectors and their labels whose \( \alpha \)'s are non-zero. The Karush-Kuhn-Tucker complementarity conditions of optimisation theory [Kar39, KT51] gives,

\[
\alpha_i [y_i (x \cdot w) + b - 1] = 0
\]

Therefore, the training data that are not on the margin (i.e., the term in square brackets is non-zero) must necessarily have a corresponding \( \alpha_i = 0 \). The training data with non-zero \( \alpha \)'s are the input vectors that lie on the edge of the margin. It is these vectors that the classifier must remember and these alone define the decision boundary. Introducing new training data outside of the margin will not change the decision boundary (so long as the new data are not in the margin or misclassified). Hence these data points are known as the support vectors. At this moment it may not be entirely clear why it is necessary to store each support vector separately when we can calculate the resultant \( w \) and just store that instead. This point will become clear later.

4.3.3 The non-separable case

Suppose the training data cannot be separated by a simple straight line (figure 4.3a). We are still restricted to using linear boundaries but the formulation above will not work. We must introduce to the classifier an ability to allow for errors and penalise them accordingly.

To achieve this, slack variables, \( \xi_i \), are introduced into the inequalities relaxing them so that some points are allowed to lie within the margin or even be misclassified. The resulting problem is to minimise the objective function,

\[
W_{\text{primal}} = \frac{1}{2} ||w||^2 + c \sum_i L(\xi_i)
\]

subject to

\[
(x_i \cdot w + b) y_i \geq 1 - \xi_i.
\]

where \( c \) is a user adjustable parameter and \( L \) is a cost function that penalises the errors made. The second term in equation (4.12) corresponds to the empirical risk.

Figure 4.3b illustrates how slack variables work. The errors are highlighted with bars drawn from the margin edge of the class in which the point belongs. The length of the bar is exactly equal to the corresponding slack variable (figure 4.3c). Only vectors that are inside the margin or are misclassified have non-zero slack variables. A non-zero slack variable allows misclassifications to satisfy the optimisation inequalities where previously they could never be satisfied. In doing so we must introduce a penalty for making errors. The contribution to the empirical risk of each error is \( \xi_i \), the length of the bars shown in
4.3. Binary linear decision boundaries

a: The data cannot be separated using a linear decision boundary.

b: We define a soft margin where some errors are allowed.

c: The errors $\xi_i$ are equal to the distance from the corresponding data point to the margin of that data point’s class.

d: The support vectors are the points that lie on the margin, are inside the margin or are misclassified.

Figure 4.3: The soft margin: adapting the SVM to work with non-separable data. (Source [Bur98].)
figure 4.3c. The support vectors, in this case, are the points that lie within or on the edge of the margin and the points that are misclassified (figure 4.3d).

### 4.3.4 Choice of cost function

The choice of the cost function on the slack variables is important and affects the solution of the SVM. There are two that can be applied such that the optimisation problem remains quadratic. They are the linear and squared cost functions. We shall review the quadratic programming objective functions for each.

#### Linear-error cost function

The linear-error cost function, \( L(\xi) = \xi \), is the most commonly used in SVMs. The primal objective is to minimise,

\[
W_{\text{primal}} = \frac{1}{2} ||w||^2 + c \sum_i \xi_i
\]  

subject to the constraints,

\[
\begin{align*}
\mathbf{x}_i \cdot \mathbf{w} + b &\geq 1 - \xi_i \quad \text{for } y_i = 1 \\
\mathbf{x}_i \cdot \mathbf{w} + b &\leq -1 + \xi_i \quad \text{for } y_i = -1 \\
\xi_i &\geq 0 \quad \forall i
\end{align*}
\]  

(4.15)

The minimisation has a more conveniently solved dual formulation that can be obtained by introducing Lagrange multipliers and a Lagrangian in a similar fashion to the separable case. The dual formulation is to maximise,

\[
W_{\text{dual}} = \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j
\]  

(4.16)

by adjusting the \( \alpha \)'s subject to

\[
\begin{align*}
0 \leq \alpha_i &\leq c \\
\sum_i \alpha_i y_i &\geq 0
\end{align*}
\]  

(4.17)

in which \( \alpha_i \) is the Lagrange multiplier of the \( i \)th constraint in the primal optimisation problem. The solution of the plane, \( \mathbf{w} \), is again given by

\[
\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i
\]  

(4.18)

Notice that the dual optimisation formulation with a linear cost function is exactly the same as the formulation in the separable case, equation (4.9), except that there is an upper limit imposed upon the Lagrange multipliers.

#### Squared-error cost function

The squared-error cost function may also be used in SVMs so that the optimisation step is a quadratic programming problem.

Applying the squared-error cost function on the slack variables, the primal optimisation problem is to minimise

\[
W_{\text{primal}} = \frac{1}{2} ||w||^2 + c \sum_i \xi_i^2
\]  

(4.19)

subject to the constraints,

\[
\begin{align*}
\mathbf{x}_i \cdot \mathbf{w} + b &\geq 1 - \xi_i \quad \text{for } y_i = 1 \\
\mathbf{x}_i \cdot \mathbf{w} + b &\leq -1 + \xi_i \quad \text{for } y_i = -1 \\
\xi_i &\geq 0 \quad \forall i
\end{align*}
\]  

(4.20)
4.4. Non-linear decision boundaries

which has an equivalent dual formulation [SSM98], to maximise

$$W_{\text{dual}} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j}^{n} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) + \frac{1}{c} \sum_{i}^{n} \alpha_i^2$$  \hspace{1cm} (4.21)

by adjusting the $\alpha_i$'s subject to,

$$\alpha_i \geq 0$$

$$\sum_i \alpha_i y_i = 0.$$  \hspace{1cm} (4.22)

Once again the solution is,

$$w = \sum \alpha_i y_i x_i.$$  \hspace{1cm} (4.23)

When the training data is quite separable and the SVM solution is sparse (i.e. the number of support vectors is small) there is very little difference between the linear and squared loss functions since most of the $\xi$'s will be zero. Only when the data is not linearly separable does the choice of loss function affect the solution.

4.4 Non-linear decision boundaries

Until now we have shown how SVMs solve linear problems only. To solve a more general non-linear problem we apply a simple mathematical trick. The same principle of maximising the margin still applies but now in higher dimensional spaces. The basics of non-linear SVMs described in this section can also be found in Burges' introduction to SVMs [Bur98].

4.4.1 Mapping to high dimensions

The trick to solving a non-linear problem using a linear boundary is to map the data using some non-linear transformation into a space with higher dimension than the input vectors [ABR64, BGV92]. To better understand this concept, let us consider the following example.

Figure 4.4a shows some sample data from the continuous exclusive OR problem. The vectors are two dimensional and exist on an $(x,y)$ plane. We shall call this plane the input space (sometimes called the data space), the space in which the input is supplied. In practice, the non-linear transformation is applied to the data only but it is important to note that the transformation applies to the whole of input space. In the example we choose to transform the space by introducing a third ($z$) component to the vectors that consists of the product of the original two. That is, a vector $\mathbf{x}$ with components $x_1$ and $x_2$ is mapped to

$$\mathbf{x}' = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix}.$$  \hspace{1cm} (4.24)

Applying this transformation to every point in input space results in the non-linear surface shown in figure 4.4c. The input space has become a manifold embedded in a three dimensional space, which we shall call the feature space. It is in the feature space that the margin is maximised, that the optimum hyperplane is found and that linear classification occurs. The effect of the transformation is to “lift” above the $z = 0$ plane all points in which both components of $\mathbf{x}$ have the same sign. All points that have different signs for $x_1$ and $x_2$ are pushed below the $z = 0$ plane. Figure 4.4d shows the data being separated by the linear boundary in feature space. The corresponding non-linear decision boundary in input space is the intersection of the $z = 0$ decision plane and the manifold, which can be seen in figure 4.4c.
4.4. Non-linear decision boundaries

a: The exclusive OR problem. This problem cannot be solved in a two dimensions using a linear boundary. The data exists in a two dimensional plane (the input space).

b: By warping the two dimensional input space as shown we map the surface into three dimensions (the feature space). The result is shown in c.

c: The exclusive OR problem can be solved in this 3D feature space by the $z = 0$ plane. The intersection of the warped surface with the decision plane is the equivalent non-linear decision boundary in the input space.

d: Viewing the transformed surface from a different angle better illustrates how the data has been separated into its two classes by a linear boundary in the feature space.

Figure 4.4: A linear decision in a high dimensional space corresponds to a non-linear decision in the input space.
4.4. Non-linear decision boundaries

a: The SVM mapping to a space of higher dimension is achieved easily. An SVM only needs to compute dot products between vectors. Take two vectors in the original 2D input space. Their dot product is \((\mathbf{u}, \mathbf{v})\).

b: Mapped to a three dimensional feature space, the same points are defined by vectors \(\mathbf{u}'\) and \(\mathbf{v}'\). In feature space the SVM requires \((\mathbf{u}', \mathbf{v}')\) to be computed. To illustrate the difference the original vectors, \(\mathbf{u}\) and \(\mathbf{v}\), are also shown as two geodesics on the surface. \((\mathbf{u}', \mathbf{v}')\) can be computed in terms of \(\mathbf{u}\) and \(\mathbf{v}\) by kernel functions. Conveniently, once a kernel function is derived no more information is required about the transformation itself.

Figure 4.5: Visualising dot products in input space and feature space.

If indeed the optimum decision boundary was the \(z = 0\) plane (which depends entirely upon the training data) the two margin edges would be the \(z = \pm 1/||\mathbf{w}_0||\) planes, where \(\mathbf{w}_0\) is the solution for the decision plane. The support vectors would be found on these two planes.

4.4.2 From dot products to kernels

In order to make non-linear decisions using SVMs it is not necessary to define an explicit transformation as we did in the example. Although it is helpful in understanding SVMs it is an unnecessary step. Analysing equations (4.9), (4.16) and (4.21) we see that the input vectors occur in the objective functions as dot products between pairs of vectors. The trick is to calculate the dot product in the feature space in terms of the input space vectors directly [ABR64]. Figure 4.5 shows how the dot products differ in input and feature spaces. Continuing with the exclusive OR example and using the notation shown in figure 4.5, the dot product in feature space is,

\[
\mathbf{u}' \cdot \mathbf{v}' = \begin{pmatrix} u_1 \\ u_2 \\ u_1 u_2 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_1 v_2 \end{pmatrix} = \mathbf{u} \cdot \mathbf{v} + u_1 u_2 v_1 v_2. \tag{4.25}
\]

Therefore, for the mapping illustrated, we can define the kernel,

\[
K(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v} + u_1 u_2 v_1 v_2. \tag{4.26}
\]
4.5. Classifying sequences using score-space kernels

We can substitute all dot products in equations (4.9), (4.16) and (4.21) with a kernel function. For example, the objective function for an SVM with a linear-error cost (4.16) becomes

\[ W_{\text{dual}} = \sum \alpha_i + \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \]  

(4.27)

which must be maximised. Given any valid kernel function we do not need to know any information about the non-linear transformation that gave rise to it. It is also significantly more efficient than performing non-linear transformations on the data and then computing their dot products separately.

The SVM output becomes

\[ f(x) = \sum_i \alpha_i y_i K(x_i, x) + b \]  

(4.28)

from which we can see the reason for storing the support vectors separately. The resultant \( w \) for the solution of the decision plane in feature space can no longer be calculated. In general, the resultant \( w \), which exists only in feature space, cannot be expressed as a non-linear transformation of a single vector in input space. However, this does not mean that the SVM solution cannot be reduced in size. It is known that the maximum number of vectors required to describe any point in a space is equal to its dimension. If the dimensionality of the feature space is lower than the number of support vectors in the solution then it is possible to find a new smaller set of input space vectors, which give the same resultant \( w \) as the support vectors when mapped to feature space. The number of vectors in the new set will equal the feature space dimensionality. Unfortunately, it is rare that the number of support vectors is greater than the feature space dimension. Some kernels can even transform data to infinite dimension spaces. However, there are approximate methods for reducing the number of support vectors in an SVM [SBS99] but these will not be discussed in this thesis.

4.4.3 General kernels

There are a few kernels that people use with SVMs regularly. A list of basic kernels is shown in table 4.1. The most commonly used are the radial basis function kernel and the polynomial kernel. Generally any function can be used so long as it represents a dot product in some space. All valid kernels must satisfy a condition called Mercer’s condition [CH53, Vap95]: for any function \( g(x) \) such that

\[ \int g(x)^2 dx \text{ is finite} \]  

(4.29)

the kernel, \( K(x, x) \) must satisfy the inequality

\[ \int K(x, y) g(x) g(y) dx dy \geq 0 \]  

(4.30)

In most cases it is not easy to check that Mercer’s condition is satisfied. One side effect is that valid kernels always generate matrices, \( K_{ij} = K(x_i, x_j) \), that are non-negative definite.

4.5 Classifying sequences using score-space kernels

The score spaces approach [SG02, SGN01], which is a generalisation of the Fisher kernel [JH99] approach, is a kernel function that enables SVMs to classify whole sequences. The score-space kernel uses any parametric generative model to achieve this. Let us introduce the score-space method in a simplified way. A variable length sequence of input vectors is mapped explicitly onto a single point in a space of fixed dimension, which is called the score-space. The mapping is achieved via the likelihood score of a
generative model by some mathematical operator. The fixed-dimension score-space allows the dot product to be computed between two sequences even if they were originally different lengths.

This section first describes a generic formulation to achieve such mappings. Afterwards we will examine in detail some special cases of the score-space method. In particular, the kernels known as the Fisher kernel [JH99] and the likelihood ratio kernel.

4.5.1 Defining the score-space

The space to which sequences are mapped is called the score-space, so named because it is defined by and derived from the likelihood score, \( p(X|M, \theta) \) of a generative model, \( M \). Given a set of \( k \) generative models, \( \{p_k(X|M_k, \theta_k)\} \), the generic formulation of the mapping of a sequence, \( X = \{x_1, \ldots, x_n\} \), to the score-space is,

\[
\psi_f^k(X) = \psi_f f(\{p_k(X|M_k, \theta_k)\})
\]

where \( f(\{p_k(X|\theta_k)\}) \), a function of the scores of a set of generative models, is called the score-argument and \( \psi_f \) is the score-mapping that maps the scalar score-argument to the score-space using the score-operator, \( \hat{F} \). The properties of the resulting score-space depends upon the choice of operator and argument that is used. Several options for score-operators were proposed by Smith et. al. [SGN01] and have been summarised in table 4.2.

Almost any function may be used as a score-argument. We shall show two specific cases that lead to the likelihood score-space kernel (more commonly known as the Fisher kernel [JH99]) and the likelihood ratio score-space kernel.

The likelihood score-space

By setting the score-argument to be the log likelihood of a single generative model, \( M \), parameterised by \( \theta \),

\[
f(\{p_k(X|M_k, \theta_k)\}) = \log P(X|M, \theta)
\]

### Table 4.1: A short list of some basic kernel functions [SSW+98].

<table>
<thead>
<tr>
<th>Name</th>
<th>( K(x, y) \equiv )</th>
<th>Parameters</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple dot product</td>
<td>( x \cdot y )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>General polynomial</td>
<td>((ax \cdot y + b)^n)</td>
<td>( a, b, n )</td>
<td></td>
</tr>
<tr>
<td>Radial basis function</td>
<td>( \exp \left( \frac{-1}{\sigma^2} (x - y)^2 \right) )</td>
<td>( \sigma )</td>
<td></td>
</tr>
<tr>
<td>Vovk’s real polynomial</td>
<td>( \frac{1}{1 - (x \cdot y)^2} )</td>
<td>( d )</td>
<td>(-1 &lt; x \cdot y &lt; 1 )</td>
</tr>
<tr>
<td>Vovk’s real infinite polynomial</td>
<td>( \frac{1}{1 - x \cdot y} )</td>
<td></td>
<td>(-1 &lt; x \cdot y &lt; 1 )</td>
</tr>
</tbody>
</table>

### Table 4.2: Some examples of score operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>first derivative</td>
<td>( \hat{F} = \nabla_0 )</td>
</tr>
<tr>
<td>first derivative and argument</td>
<td>( \hat{F} = \lbrack \nabla_0, 1 \rbrack^T )</td>
</tr>
<tr>
<td>first and second derivative</td>
<td>( \hat{F} = \lbrack \nabla_0, \text{vec}(\nabla^2 \theta)^T \rbrack^T )</td>
</tr>
</tbody>
</table>
4.5. Classifying sequences using score-space kernels

and choosing the first derivative score-operator from table 4.2 we obtain the mapping for the likelihood score space.

\[
\psi(X) = \nabla_{\theta} \log P(X|M, \theta)
\]  

(4.33)

This mapping is also known as the Fisher mapping and was first developed and applied successfully for biological sequence analysis by Jaakkola and Haussler [JH99]. It has also been applied to speaker identification by Fine et al. [FNG01] and in web audio classification by Moreno and Rifkin [MR00].

Each component of the score-space space, \(\psi(X)\), corresponds to the derivative of the log likelihood score with respect to one of the parameters of the model. In some ways it is a measure of how well the sequence, \(X\), matches the model. Consider a generative model being trained using maximum likelihood criterion and gradient descent. In order to maximise the likelihood of a given sequence the same set of derivatives to equation (4.33) must be computed so that the parameters may be updated. When the derivatives are small then the likelihood has reached a local maximum. Likewise, when the derivatives are large then the likelihood has yet to reach a maximum. Whether the derivatives will provide additional information that is not already encoded in the likelihood score may be examined by the following extension to the kernel. The converse may also be examined.

Using the first derivative with argument score-operator and the same score-argument the mapping becomes

\[
\psi(X) = \nabla_{\theta} \log P(X|M_1, \theta)
\]  

(4.34)

The score-space space defined by this mapping is identical to the Fisher mapping with one extra dimension which consists of the log likelihood score itself. This mapping has the benefit that the performance of a classifier using these mappings will have a minimum test performance that equals the original generative model, \(M\). The inclusion of the derivatives as “extra features” should give additional information for the classifier to use.

The likelihood ratio score-space

An alternative score-argument is the ratio of two generative models, \(M_1\) and \(M_2\),

\[
\psi(X) = \nabla_{\theta} \log \frac{P(X|M_1, \theta)}{P(X|M_2, \theta)}
\]  

(4.35)

where \(\theta \equiv [\theta_1, \theta_2]\). The corresponding mapping using the first derivative score-operator is,

\[
\psi(X) = \nabla_{\theta} \log \frac{P(X|M_1, \theta_1)}{P(X|M_2, \theta_2)}
\]  

(4.36)

and using the first derivative with argument score-operator,

\[
\psi(X) = \left[ \frac{\log P(X|M_1, \theta_1)}{\nabla_{\theta} \log \frac{P(X|M_1, \theta_1)}{P(X|M_2, \theta_2)}} \right].
\]  

(4.37)

The likelihood ratio score-space is motivated by the likelihood ratio GMM (GMM-LR) classifier described in section 3.2.2. In the same way that the GMM-LR is a more discriminative classifier than a single GMM alone, then so too should the corresponding score-space kernel. A GMM likelihood ratio forces the classifier to model the class boundaries more accurately. The discrimination information encoded in the likelihood ratio score should also be in its derivatives.
4.5.2 Computing the score-space vectors

In this section we derive the formulae for the computing derivatives of the log likelihoods when the generative model is a diagonal covariance GMM. The formulae for the derivatives when the generative model is an HMM may be found in [SGN01].

The diagonal covariance GMM likelihood is written as follows,

$$P(x|M_2, \theta) = \sum_{j=1}^{N_t} a_j \prod_{i=1}^{N_k} \frac{1}{\sigma_j \sqrt{2\pi}} \exp \left\{ -\frac{1}{2} \left( \frac{x_i - \mu_{ij}}{\sigma_j} \right)^2 \right\}$$  \hspace{1cm} (4.38)

where $\theta = \{ a_j, \mu_{ij}, \sigma_j^2 \}$ which is the set of parameters in the GMM, $M$. In particular, $a_j$ is the prior of the $j^{th}$ component of the model, $\mu_j$ is the mean vector of the $j^{th}$ component and $\sigma_j$ is the diagonal covariance vector. The superscript on the mean and covariance enumerate the components of the vectors. $N_k$ is the number of Gaussians that make up the mixture model and $N_d$ is the dimensionality of the input vectors with components $x = [x_1, x_2, \ldots, x_N]^T$.

Define the global likelihood of a sequence $X = \{x_1, \ldots, x_N\}$ as

$$P(X|M_2, \theta) = \prod_{i=1}^{N_t} P(x_i|M_2, \theta)$$  \hspace{1cm} (4.39)

where $N_t$ is the number of vectors in the sequence. A fixed length vector may be constructed in a high dimensional space by computing the vector of the derivatives of the log of (4.39) with respect to each of the parameters. The derivatives are with respect to the covariances, means and priors of the Gaussian mixture model. Let

$$R(i, j) = \sum_{k=1}^{N_j} \frac{1}{\sigma_j \sqrt{2\pi}} \exp \left\{ -\frac{1}{2} \left( \frac{x_i - \mu_{ij}}{\sigma_j} \right)^2 \right\}.$$  \hspace{1cm} (4.40)

The derivative with respect to the $j^{th}$ prior is,

$$\frac{d}{da_j} \log P(X|M_2, \theta) = \sum_{i=1}^{N_t} \frac{R(i, \mu_j)}{\sum_{j=1}^{N_j} a_j R(i, j)}$$  \hspace{1cm} (4.41)

The derivative with respect to the $k^{th}$ component of the $j^{th}$ mean is,

$$\frac{d}{d\mu_{ij}} \log P(X|M_2, \theta) = -\sum_{i=1}^{N_t} \frac{a_j R(i, \mu_j)}{\sum_{j=1}^{N_j} a_j R(i, j)} \cdot \frac{1}{\sigma_j^2} \left( \frac{x_i - \mu_{ij}}{\sigma_j^2} \right)$$  \hspace{1cm} (4.42)

Lastly, the derivative with respect to the $k^{th}$ component of the $j^{th}$ covariance is,

$$\frac{d}{d\sigma_{ij}^2} \log P(X|M_2, \theta) = -\sum_{i=1}^{N_t} \frac{a_j R(i, \mu_j)}{\sum_{j=1}^{N_j} a_j R(i, j)} \cdot \frac{1}{\sigma_{ij}^4} \left( \frac{x_i - \mu_{ij}}{\sigma_{ij}^2} \right)^2$$  \hspace{1cm} (4.43)

The fixed length score-space vector can then be expressed as,

$$\Psi_\theta(X) = \left[ \frac{d}{da_j}, \ldots, \frac{d}{d\mu_{ij}}, \ldots, \frac{d}{d\sigma_{ij}^2} \right]^T \log P(X|M_2, \theta)$$  \hspace{1cm} (4.44)

for all $j^*$ and $k^*$, where $j^*$ runs from one to the number of Gaussians in the mixture model, $N_k$, and $k^*$ from one to the dimensionality of the input vectors, $N_d$. 

---

47
The likelihood ratio kernel is also computed using equations (4.41) to (4.44). The fixed length vectors of the likelihood ratio kernel (4.36) can be expressed as the difference of two terms,

\[ \psi_0(X) = \nabla_0 \log P(X|M_1, \theta_1) - \nabla_0 \log P(X|M_2, \theta_2) \]  

(4.45)

Let \( \theta = \{\theta_1, \theta_2\} \) be the vector of all parameters that exist in both models, \( M_1 \) and \( M_2 \). The derivatives of \( \log P(X|M_1, \theta_1) \) with respect to the parameters \( \theta_2 \) in \( M_2 \) is zero and vice-versa. Thus \( \psi(X) \) for the likelihood ratio kernel can be split so that the derivatives are computed with respect to one model at a time. When the differentiated parameter belongs to model \( M_1 \) then

\[ \psi_0_1(X) = \nabla_0 \log P(X|M_1, \theta_1) \]  

(4.46)

is computed. Likewise, when the parameter belongs to model \( M_2 \) then

\[ \psi_0_2(X) = -\nabla_0 \log P(X|M_2, \theta_2) \]  

(4.47)

is computed. The reader will notice that these derivatives are identical to the derivatives computed by the Fisher kernel. The fixed length vectors of the likelihood ratio kernel is,

\[ \psi_0(X) = \begin{bmatrix} \psi_0_1(X) \\ \psi_0_2(X) \end{bmatrix} \]  

(4.48)

From equations (4.44) and (4.48) it can be seen that the dimensionality of the score-space is equal to the total number of parameters in the generative models. Having mapped the sequence to the score-space any discriminative classifier may be used to classify vectors and hence obtain a classification for the complete sequence. However, it is not unusual for generative models to have several thousand parameters. This means that the discriminative classifier must be able to classify vectors of that size. Classifiers such as MLPs cannot be easily trained on such data due to problems in parameterisation. The SVM, by its nature, is ideally suited to classify high dimensional data.

### 4.5.3 Dot products in score-space

For a given score-space, the metric of the space is defined by the generative models and is generally non-Euclidean. To correctly compute the dot product in any space requires the local Riemannian metric at each point in the space. We shall assume that the metric is the same throughout the whole of score space. A kernel constructed from any of the above mappings is,

\[ K(X,Y) = \psi(X)^T G \psi(Y) \]  

(4.49)

where \( G \) defines the metric of the space. In Euclidean space \( G \) is the identity matrix. In the case of the log likelihood score-space mapping, \( G \) is the inverse Fisher information matrix,

\[ G = (E(\psi(X)\psi(Y)^T))^{-1}. \]  

(4.50)

assuming that \( E(\psi(X)) = E(\psi(Y)) = 0 \). To put it another way, \( G \) is the inverse of the covariance matrix of the vectors in score-space.

This can be interpreted as a whitening step where the score-space components are normalised to zero mean and unit variance. It is easy to see why whitening is an important step. Consider a two dimensional space where the variance in one dimension is significantly higher than in the other. A dot product in this case will be dominated by the high variance component, reducing the dimensionality of the space to one
and losing the information contained in the other dimension. It is especially important to whiten the data when applying SVMs: SVMs are not invariant to linear transformations in the feature space (see section 5.2 and figure 5.1).

To correctly compute dot products in score-space relies on the ability to compute a full covariance matrix, which will not only normalise the scaling in each dimension but also make the principal component axes of the space orthogonal. However, the score-space space dimensionality, being equal to the number of parameters in the generative model, may be large. If there were $10^5$ parameters in the generative model (a small number by many standards) then the full covariance matrix would have $10^{10}$ elements. A covariance matrix of this size is impractical to compute and invert. At the very least the data should be normalised by the diagonal covariance matrix so that the scale of each dimension is the same. Better is a block diagonal covariance matrix to make some of the principal component axes orthogonal.

### 4.6 Optimising SVMs

The dual formulation of the SVMs described can all be solved by quadratic programming. There are numerous methods for solving a quadratic programming problem [Lue84] and there are many commercial optimisers available that do the job efficiently [Van94, MS98].

One of the difficulties of the quadratic problem is that memory requirements for the objective function grow as the square of the size of the training set. The term

$$ y_i y_j a_i a_j K(x_i, x_j) $$

(4.51)

in equations (4.9), (4.16) and (4.21) represents a square matrix with $N^2$ elements. This means that a quadratic programming optimiser can realistically train SVMs on approximately 5000 input vectors at most (which, assuming 64 bit double precision numbers, would require over 200 megabytes of memory to store the objective function).

Fortunately with SVMs, it is not necessary to solve the optimisation problem for the entire data set in a single step. The definition of the support vectors is that they alone define the solution for the decision boundary. Any other correctly classified data that lie outside the margin will not become part of the solution and may be discarded. Taking advantage of this fact the SVM can be trained in multiple steps. A large optimisation problem can be split into many smaller optimisation problems. First solve the problem on a small manageable subset (a chunk) of the training data to get one set of support vectors. Adding to these support vectors another subset of the data will yield a slightly better solution that will discard some of the old support vectors but include some new ones. This is repeated until all the training data is seen. The final solution will be the same as optimising in a single step.

The method of splitting the data training is called chunking. The efficiency of chunking varies significantly depending upon its implementation. If a data point is already correctly classified it will not change the solution therefore it is unnecessary to add that point to the next iteration. Osuna [OFG97] showed that SVMs may be optimised without including, in the current training chunk, the support vectors whose Lagrange multipliers have already reached their upper bound. Indeed, each chunk need not contain all of the support vectors. A method called sequential minimal optimisation [Pla99] reduces the chunk size down to just two vectors at a time. This removes the dependency upon an efficient quadratic programming optimiser since the quadratic program for two examples can be solved analytically. Flake [Pla99] improved sequential minimal optimisation by changing the rules for selecting which training vectors to optimise next. By choosing the heuristics carefully it is possible to make chunking very efficient.
In general, the use of a good chunking algorithm is essential when training on large databases. Although there will be no effect on the final solution, a poor chunking algorithm will take longer to converge.

## 4.7 Properties of SVMs

### 4.7.1 Regularised risk

Let us return to the regularised risk introduced in equation (4.2).

\[ R_{reg}(w) = T(w) + cR_{emp}(w) \]  \hspace{1cm} (4.52)

Compare the regularised risk formulation with the SVM’s primal objective, which is to minimise

\[ W_{primal} = \frac{1}{2}||w||^2 + c \sum_{i} L(\xi_i). \]  \hspace{1cm} (4.53)

The first term in (4.53) results from maximising the margin and may be seen to be penalising the SVM’s complexity. The second term is the empirical risk, which is an accumulation of all the training errors made by the SVM. In this way, the SVM has a form of built-in regularisation. This contrasts with other classifiers, such as MLPs, in which regularisation may be included as part of the optimisation but is not built into the objective function.

One benefit of the SVM’s regularised objective function is that it is convex. This means that the quadratic programming problem always has a global minimum that occurs when the margin is maximised. Therefore, when an SVM has been trained we can be certain that the global optimum is reached if there are no training samples inside the margin. For objective functions derived from other criteria that do not yield convex objective functions, such as using the empirical risk only or using maximum likelihood, a global minimum cannot be guaranteed, instead a local optimum is usually reached.

### 4.7.2 Comparison with RBFs

With an appropriate choice of kernel, SVMs can be made to look like other classifiers. For example, the RBF kernel gives rise to a radial basis function (RBF) network type solution. A more complete comparison between SVMs and RBF networks may be found in [SSB+97]. Here we shall make an analytic comparison using a simple toy problem.

An RBF network consists of a GMM in its first layer (figure 4.6) trained using the EM algorithm to model the probability distribution of the data. The second layer is a linear combination of the outputs of the individual Gaussian components in the GMM. The output of an RBF network with a single output unit is,

\[ f(x) = \sum_{i} a_i \exp \left( -\frac{1}{2}(x-\mu_i)^T \Sigma_i (x-\mu_i) \right) \]  \hspace{1cm} (4.54)

where \( \mu_i \) and \( \Sigma_i \) are the mean and covariance of the \( i \)th Gaussian respectively and \( a_i \) is the corresponding network weight. Using radial Gaussians, the RBF network becomes,

\[ f(x) = \sum_{i} a_i \exp \left( -\frac{(x-\mu_i)^2}{2\sigma_i^2} \right) \]  \hspace{1cm} (4.55)

The output of an SVM with the RBF kernel is,

\[ f(x) = \sum_{i} y_i a_i \exp \left( -\frac{(x-x_i)^2}{2\sigma} \right), \]  \hspace{1cm} (4.56)
One can immediately see the similarity between the output functions of the two classifiers. The weights in the network correspond to the Lagrange multipliers times the class label and the Gaussian means correspond to the support vectors. Since the two classifiers are trained differently the interpretation of the variables are different. Figure 4.7 shows a one dimensional example to highlight to difference between RBF networks and SVMs.

The SVM solution is shown in figure 4.7a. Since the one dimensional problem is separable all points must lie outside of the margin except the support vectors which must lie on the margin. Given a suitably chosen width parameter for the RBF kernel the only way in which the margin conditions are met is to have the four points at the class edges as the support vectors. The four support vectors become the basis functions for the solution shown. The SVM automatically determines the number of basis functions in its solution to achieve maximum discrimination.

In contrast, conventional RBF network training requires the user to choose the number of Gaussian components in the GMM. However, conventional training allows the Gaussian widths to be adapted independently of each other while the SVM keeps them all the same. If a two component GMM is trained using the EM algorithm then the RBF network would find the solution shown in figure 4.7b. The second Gaussian is drawn upside-down because it would have a negative \( a_i \) coefficient so that all squares are classified positive and triangles negative. The solution of a four mixture RBF network is illustrated in figure 4.7c. It is slightly closer to the SVM solution but the Gaussian means do not match the support vectors since the optimisation criteria are different.

A similar comparison may be performed with the tanh kernel and a two-layer neural network with tanh activations at the hidden units and a linear activation at the output. Chapter 5 compares an SVM with the polynomial kernel to a polynomial classifier in detail.

### 4.7.3 SVM extensions for more than two classes

The SVM formulation described can only solve binary problems. The extension of SVMs to problems that are not binary is important. There have been many attempts to combine binary classifiers, including SVMs in particular, to solve problems containing more than two classes [Sch97, GMM98, Kre99, MA99, FF98, Fri96, MM98]. There have also been an attempts to incorporate the classification of more than two classes into the optimisation process of the SVM so that all classes are processed simultaneously [AC00, WW98]. Here is a short summary of the more commonly used methods.
4.7. Properties of SVMs

a: We get the SVM solution by locating the support vectors. Since the support vectors themselves must have an output of either +1 or −1 and all other points must lie outside the margin (−1 ≤ classifier output ≤ +1) then the support vectors must be the four points at the edges of the classes. Assuming a particular RBF kernel parameter the support vectors give rise to the four basis functions (dotted lines). Summing the basis functions gives the SVM solution (solid line).

b: The RBF network solution is obtained by modelling the data using a two component GMM. The output node of the RBF network subtracts the Gaussian components (dotted lines) to obtain the classifier output.

c: The RBF network solution using a four component GMM is more similar to the SVM solution. The difference between the support vectors and the location of the Gaussian means are visible. Furthermore, the width of the Gaussians are learned while the basis function widths on the support vectors cannot be adapted automatically.

Figure 4.7: A simple comparison between RBF networks and an SVM with the RBF kernel.
Table 4.3: An example of error correcting codes

<table>
<thead>
<tr>
<th>Classifier</th>
<th>A</th>
<th>B</th>
<th>infer class</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>outputs</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>outputs</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

Each node of the tree contains a classifier that is trained to distinguish between two classes only. Each node is therefore an expert in only two of the four classes shown. No individual node can say whether a data vector belongs to a particular class. However, each node can confidently say which class the vector does not belong. One class is eliminated at each level of the tree until only one class remains.

Figure 4.8: Directed acyclic graph for classifying four classes using binary classifiers

1. **One from many**
   This solution involves training as many classifiers as there are classes. Each classifier discriminates one class from all other classes and the classifier with the highest output is chosen. This was one of the methods used by Schmidt and Gish [SG96] for speaker identification.

2. **Error correcting codes**
   This method involves a number of classifiers each performing a different split in the data. A particular combination of classification by each classifier leads to the identification of one particular class [MM97, GMM98]. For example, in table 4.3 a four class problem is solved using two classifiers.

3. **Decision trees**
   In a decision tree each node of the tree makes a decision on whether a particular data point belongs to a particular subset of classes. Subsequent nodes in the tree further reduce the number of classes until only one class is left. Directed acyclic graphs [PCST00] are a good example (figure 4.8) that have been used successfully with SVMs.
4.8 Concluding remarks

The SVM is a binary classifier based on a linear discriminant. Due to the way the optimisation problem has been formulated, the SVM is able to make non-linear decisions through the use of kernel functions. By choosing an appropriate kernel it is possible for the SVM to mimic other discriminative classifiers such as RBF networks and polynomial classifiers.

The theoretical interest of the SVM and the maximum margin idea has inspired a large number of SVM variants including the relevance vector machine [Tip99] and generalised SVM by Mangasarian [Man99]. The original SVM formulation still remains popular and there are many public domain SVM trainers.

SVMs have been applied to a variety of different tasks. These include written digit recognition by Cortes and Vapnik [CV95] and face recognition by Osuna [OFG97]. Small scale applications to speech data classification using SVMs with polynomial and RBF kernels include phone recognition by Ganapathiraju et al. [GHP98], vowel recognition by Clarkson and Moreno [CM99] and stop consonant detection by Niyogi et al. [NBR99]. More recently, Ganapathiraju et al. [GHP02] used SVMs (with a linear cost function and a sigmoid) to estimate the emission probabilities for HMMs on a large vocabulary task and reported results that are better than baseline HMMs. There have also been applications to speaker recognition tasks [SG96, FNG01, BM01, QB02] also using polynomial, RBF and Fisher kernels. We will be investigating the applicability of the SVM to speaker verification.
Chapter 5

Developing speaker verification SVMs on YOHO

5.1 Introduction

The RBF network, MLP, GMM, polynomial classifier and SVM are static classifiers. A static classifier is one that does not directly classify sequential data where the length of the sequence is variable. When applying static classifiers to speaker verification one must find a method that allows variable length sequences to be classified. A commonly used strategy is to score the frames of an utterance individually and combine the results afterwards by taking their mean. This approach scores utterances in a way that is independent of the specific order of the vectors in the sequence: a time independent approach. The review of RBF networks, MLPs, GMMs and polynomial classifiers in chapter 3 described this approach to sequence scoring for these classifiers.

Schmidt and Gish [SG96] reported promising SVM results on a speaker identification task using the time independent approach described above. The SVM assigns a score, which we call the frame score, to each frame of an utterance. A score for the whole utterance, which we call the utterance score, is computed by taking the mean of the frame scores.

A different sequence scoring strategy available to the SVM is the score-spaces approach [JH99, SG02, SGN01] described in section 4.5. This approach exploits a generative model to obtain a fixed-length score-space vector (or score vector) that represents one entire sequence. The mean of the frame scores is just one method of computing utterance scores. The score-space approach uses a set of different methods for computing a set of scores that are each analogous, in the way they are computed, to the utterance score described above. The score-space approach is appropriately named since each component of the score-space vector is analogous to an utterance score (although we shall not refer to them as utterance scores, instead they shall be collectively known as the score vector). The approach may be interpreted as a way of mapping the variable length sequences to fixed length score vectors. The score vectors may then be classified by the SVM to obtain the utterance score.

Each component of the score vector corresponds to a derivative of the log-likelihood of the generative model with respect to one of the model parameters. If the log-likelihood is time independent then the score vector components will also be time independent. This is reflected in the equations for computing the score-space vectors in section 4.5.2. This is significant since we are primarily concerned with text independent speaker verification. Only if we were concerned with text dependent speaker verification...
should the score-space components be time dependent: by knowing the exact phrase spoken is it possible to model the temporal dependencies in an utterance.

### 5.1.1 Layout of this chapter

This chapter describes the development work that was done in applying SVMs to speaker verification. The YOHO database described in section 2.4.1 was used. Section 3.3.2 noted that a polynomial classifier is similar to an SVM with a polynomial kernel. In section 5.2 we compare these two in more detail. Polynomial classifiers provide a good baseline for the YOHO experiments. Their similarity with SVMs allows us to separate the model from the method. Given that the polynomial classifier is known to perform well on YOHO and since the functional form of the decision boundaries made by these two classifiers are identical, then we expect that the SVM will perform as well as the polynomial classifier. It is reasonable to aim to achieve a polynomial SVM performance that, as a minimum, equals the polynomial classifier. In the initial experiments the SVMs did not reach the target performance. This implied that factors other than the shape of the decision boundary had to be examined. Simulations were performed using SVMs that were each trained slightly differently to observe the effect of the various factors that may affect performance. Factors investigated include the order of the polynomial, whether data normalisation was required and the type of cost function applied to the slack variables.

Section 5.3 describes a novel technique called spherical normalisation. The new technique is used to normalise the polynomial kernel enabling the use of higher order polynomials. We describe some interesting properties that arise from applying the normalisation such as imparting RBF-like properties on the kernel. Spherical normalisation also enables us to better understand the effects of assigning certain values to the parameters of the general polynomial kernel. The normalisation may also be generalised so that it may be used to normalise any valid kernel. Spherically normalised polynomial kernel SVMs are evaluated on the YOHO speaker verification task yielding significant reductions in the average equal error rates compared to the unnormalised polynomial kernel SVM. It was found that the spherically normalised polynomial kernel SVM performance matches that of the polynomial classifier.

Section 5.4.2 describes the simulations performed using the score-space approach to variable length sequence classification. The performance of score-space approach using only the client model immediately matched the performance of the polynomial classifier. It is also significantly better than the corresponding GMM baseline that uses only the client model. The score-space approach involving client and world models was not tested on the YOHO database. This is because the performance of the baseline GMM classifier that uses client and world models could not be measured reliably on YOHO: on the test data the GMM-LR baseline could achieve 0.00% average equal error rate. Any performance gain by the corresponding score-space approach would not be observable.

The results of this chapter have been published in [WC00, WR02].

### 5.1.2 Experimental procedure

The YOHO database, summarised in section 2.4.1, does not specify with sufficient detail which utterances should be used for training and testing. Although each speaker has 95 training and 40 testing utterances there is no specification as to which speakers should be trained against each client and which speakers should be used only during testing as unseen impostors. We therefore define our own training and testing protocol for use with all experiments involving YOHO. All the simulations performed on the YOHO database adhered to the following experimental training and testing protocol strictly.
5.2 SVMs vs. polynomial classifiers

In this section we shall refer to the SVM with polynomial kernel as the SVM. The underlying principles that enable non-linear classification in the SVM and the polynomial classifier are similar. Let us compare the properties of the two classifiers. Table 5.1 summarises the results of this section, listing performance of the baseline polynomial classifier and various SVMs, each differently trained to observe the effect of a particular modification. The results for the polynomial classifier were gratefully received from Campbell using the polynomial classifier described in section 3.3.2 and [CTB00]. The polynomial classifier results were obtained using the protocol described in section 5.1.2.

Prior to classifying by a linear discriminant, the data is mapped to a feature space. This mapping is made by a polynomial expansion: implicitly by the SVM and explicitly in the polynomial classifier. The similarity between the two methods can be seen easily. An explicit expansion of a two dimensional vector is,

\[
x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \Rightarrow \quad \phi(x) = \begin{bmatrix} 1 & x_1 & x_2 & x_1 x_2 & x_1^2 & x_2^2 \end{bmatrix}^T
\]  

(5.1)
Table 5.1: SVM vs. polynomial classifiers: summary of results. The polynomial classifier results were kindly provided by Campbell [CTB00]. SVMs were trained, each time modifying a different property to determine its effect on the classifier’s performance. The test reported figures are the average test EERs over all the speakers in the database.

<table>
<thead>
<tr>
<th>Classifier type</th>
<th>poly. order</th>
<th>quant-isation</th>
<th>error function</th>
<th>data norm</th>
<th>training set impostors avg. eer (%)</th>
<th>unseen impostors avg. eer (%)</th>
<th>avg. number SV’s</th>
</tr>
</thead>
<tbody>
<tr>
<td>polynomial classifier</td>
<td>2</td>
<td>none</td>
<td>squared</td>
<td>no</td>
<td>1.30</td>
<td>1.65</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>none</td>
<td>squared</td>
<td>no</td>
<td>0.18</td>
<td>0.38</td>
<td>1</td>
</tr>
<tr>
<td>SVM</td>
<td>2</td>
<td>100</td>
<td>linear</td>
<td>no</td>
<td>5.00</td>
<td>5.60</td>
<td>266</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>100</td>
<td>linear</td>
<td>squared</td>
<td>3.35</td>
<td>3.76</td>
<td>938</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>100</td>
<td>linear</td>
<td>yes</td>
<td>2.94</td>
<td>3.37</td>
<td>302</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>100</td>
<td>linear</td>
<td>no</td>
<td>1.43</td>
<td>1.73</td>
<td>377</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>200</td>
<td>linear</td>
<td>no</td>
<td>0.98</td>
<td>1.30</td>
<td>650</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>100</td>
<td>squared</td>
<td>no</td>
<td>1.36</td>
<td>1.75</td>
<td>742</td>
</tr>
</tbody>
</table>

The second order polynomial kernel for two dimensional data is,

\[
K(x,y) = (x \cdot y + 1)^2 
\]  \hspace{1cm} (5.2)

\[
= (x_1y_1 + x_2y_2 + 1)^2 
\]  \hspace{1cm} (5.3)

\[
= \begin{bmatrix} 1 \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \sqrt{2}x_1x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ \sqrt{2}y_1 \\ \sqrt{2}y_2 \\ \sqrt{2}y_1y_2 \\ y_1^2 \\ y_2^2 \end{bmatrix} 
\]  \hspace{1cm} (5.4)

Hence, with the exception of the coefficients, the expansion terms are identical.

The explicit expansion of each input vector into feature space is computationally expensive, increasing with polynomial order and input vector dimensionality. Table 5.2 shows the number of terms in an expansion of 24 and 25 dimensional input data for various polynomial orders. Low order polynomial expansions are manageable but, for high order expansions, the first problem will be to store the expanded feature vectors of the entire training set. The SVM is more efficient. By computing the value of the dot product in terms of the input vectors there is no explicit expansion and therefore no limit on the polynomial order. However, there is a practical limit. Increasing the order increases the feature space dimensionality which, in turn, increases the number of support vectors required to define the decision hyperplane. There is also a numerical limitation, which occurs when the kernel values are very large or very small.

Supposing that storage space is infinite, the dimensionality of the feature space would be so large that the polynomial classifier, which minimises the squared error during optimisation, would become over-trained. For example, using a tenth order polynomial expansion and given all the data available in YOHO, a linear discriminant in 131 million dimensional space is certain to over-fit. However, SVMs are especially suited to these situations. As a result of the maximum margin criterion, the SVM is a regularised classifier and will not over-fit data in high dimensional spaces. The regularisation parameter was set to 10 in the
Table 5.2: Size of feature space as a function of polynomial order

<table>
<thead>
<tr>
<th>Polynomial order</th>
<th>24D input space</th>
<th>25D input space</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>325</td>
<td>351</td>
</tr>
<tr>
<td>3</td>
<td>2 925</td>
<td>3 276</td>
</tr>
<tr>
<td>4</td>
<td>20 475</td>
<td>23 751</td>
</tr>
<tr>
<td>6</td>
<td>593 775</td>
<td>736 281</td>
</tr>
<tr>
<td>8</td>
<td>10 518 300</td>
<td>13 884 156</td>
</tr>
<tr>
<td>10</td>
<td>131 128 140</td>
<td>183 579 396</td>
</tr>
</tbody>
</table>

comparison experiments of this section. This value was roughly estimated by crossvalidation using two
speakers. To obtain a better estimate all of the speakers should have been used but this was not done due
to time and resource limitations.

The YOHO data set consists of approximately 20,000 feature vectors per speaker. Since the data is
noisy, the SVMs will retain a significant number of frames as support vectors. This makes the SVM im-
practical to use compared to the polynomial classifier, which has a decision boundary that is parameterised
by the equivalent of one support vector and handles large quantities of data well. To work around this
problem a vector quantisation method was used to compress the training data.

The k-means clustering algorithm was used to obtain 100 codebook vectors to represent the training
data of each speaker. Training according to the protocol defined in section 5.1.2 meant that each SVM was
trained on 6900 codebook vectors. Here we trained SVMs using a linear cost function applied to the slack
variables. The test results for second and fourth order SVMs are 5.60% and 1.73% respectively (lines 3
and 6 in table 5.1). By doubling the number of codebook vectors the fourth order SVM yielded 1.30%
EER (table 5.1 line 7). In comparison, the performance of the second and third order polynomial classifiers
trained on unquantised data are 1.65% and 0.38% EER (table 5.1 lines 1 and 2). The accuracy lost due
to vector quantisation is expected. Higher than fourth order SVMs could not be trained due to numerical
limitations that prevented the optimiser from converging.

One of the fundamental differences between the polynomial classifier and the SVM in the simulations
above is the way in which errors are penalised. The polynomial classifier penalises all deviations from the
desired output using a squared error cost function. In the SVM experiments above a linear cost function
was used and this penalises only the points that are misclassified or lie inside the margin. The asymmetry
by which the SVM penalises deviations from the desired output is the fundamental property of the SVM
that allows it to make sparse solutions. That cannot be changed. However, we can match the cost function
to that of the polynomial classifier by using the squared error cost function.

Using 100 codebook vectors per speaker and the squared error cost function, the second and fourth
order SVMs yielded 3.76% and 1.75% EER respectively (table 5.1 lines 4 and 8). A significant reduction
from 5.6% for the second order SVM but no change for the fourth. This is not surprising. The fourth order
SVM maps data to a twenty thousand dimensional space. Since there are only 6900 training vectors then
the training data must be linearly separable in the feature space. When the problem is linearly separable
then all support vectors lie on the margin and the slack variables are not required so the SVM solution is
independent of the cost function.

Another important difference between the SVM and the polynomial classifier is that the latter is invari-
ant to linear transformations while the former is not. This means that training an SVM on data that has a large variance in one component will give a different solution from when that component is normalised. Training the polynomial classifier in both cases will yield the same result.

When mapping from input to feature space using a polynomial expansion, any linear transformation on \( x \) translates to the feature space as a linear transformation on \( \phi(x) \) as long as all the terms of the polynomial expansion are present. This applies whether using the kernel or an explicit expansion. Consider the example of the polynomial kernel for two dimensional data in equation (5.2). Apply a transformation, \( L \), to the data,

\[
K(Lx, Ly) = (Lx \cdot Ly + 1)^2.
\]

Expand the transformation matrix \( L = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \).

\[
K(Lx, Ly) = ((ax_1 + bx_2)(ay_1 + by_2) + (cx_1 + dx_2)(cy_1 + dy_2) + 1)^2.
\]

By collecting the common terms it can be seen that the feature space vectors may be expressed as a linear transformation of the vectors in equation (5.4).

The polynomial classifier’s invariance to linear transformations can be seen easily. Starting from equation (3.25),

\[
M^T Mw = M^T o,
\]

apply some linear transformation, \( L \), to the data in feature space, \( M = NL \),

\[
L^T N^T NLw = L^T N^T o = N^T o
\]

Thus applying any linear transformation to the data changes the solution by the same transformation.

To illustrate that the SVM is not invariant to linear transformations consider figure 5.1. Figure 5.1b is a stretched version of the 5.1a. After maximising the margins, the two solutions have different numbers of support vectors. Therefore, the two solutions cannot be related by the same transformation that was applied to the data.

Data normalisation should take place in the feature space where the margin is maximised. However, it is not clear how this can be achieved through the kernel. Instead, the data shall be normalised to zero mean and unit variance in the input space. One hundred codebook vectors were recomputed from the normalised data and a second order SVM trained. This yielded 3.37% EER (table 5.1 line 5), a significant reduction from 5.6%. The optimiser could not converge to a solution for the fourth order SVM.

Although the decision boundaries are both polynomials, the fundamental differences between the polynomial classifier and the SVM are significant. The SVM appears to be at a disadvantage due to its reliance on vector quantisation and manual data normalisation. In its favour, it has a regularised cost function and an efficient way of mapping data to high dimensions. However, neither of these properties have been exploited properly since the higher order SVMs could not be trained. Only by using a high order SVM are the vectors mapped to a high dimensional space where the data are linearly separable so that the margin may be maximised. The results described in this comparison are summarised in table 5.1.

5.3 Spherically normalised polynomial kernels

SVMs are formulated in a way that enables them to perform well under sparse data conditions, where other classifiers may struggle. However, SVMs may struggle when the data is not sparse, especially when the
data is highly inseparable. The best way to train an SVM is to map to a feature space where the data are sparse and a linear boundary will separate the classes. This can be achieved by mapping the input vectors to a sufficiently high dimensional space, one whose dimension exceeds the size of the training set. For the polynomial kernel this corresponds to using a larger value of $n$ (see table 5.2).

In the previous section data normalisation yielded the single largest improvement in performance for a second order kernel but simultaneously rendered the fourth order kernel untrainable. The source of the problem is in the kernel scaling. The term $(\mathbf{x}, \mathbf{y})$ in the polynomial kernel need only increase by a small amount for the value of the kernel to increase significantly. This is especially true if $n$ is large. The Hessian computed from this kernel is likely to be ill-conditioned and sometimes singular. An attempt to alleviate the problem by rescaling the input vectors by a scalar has little effect. The scaling factor applied to the data may be factored out as a constant in front of the Hessian, thus leaving it unchanged.

An alternative method is to normalise the lengths of the feature vectors. Let $\hat{\mathbf{x}}$ denote a vector, $\mathbf{x}$, normalised to unit length. This constrains the dot products to the range $-1 \leq \hat{\mathbf{x}} \cdot \hat{\mathbf{y}} \leq +1$ since the dot product of two unit vectors is the cosine of the angle between them. The poor scaling due to large powers of $n$ may be further alleviated by applying an affine transform so that the polynomial kernel becomes,

$$K(\mathbf{x}, \mathbf{y}) = \left( \frac{\hat{\mathbf{x}} \cdot \hat{\mathbf{y}} + 1}{2} \right)^n \quad (5.10)$$

The output of this kernel function is ranged between zero and one for all $n$.

Vector length normalisation is most easily achieved by dividing a vector by its Euclidean length. However, this results in information loss and greater classification uncertainty. For example, two points in input space represented by $\mathbf{x}$ and $2\mathbf{x}$ will both normalise to $\hat{\mathbf{x}}$. Alternatively, normalisation may be done in a higher dimensional space without information loss. Consider the mapping from a 2D plane to a 3D unit sphere illustrated in figure 5.2. Any point in 2D space may be mapped onto the surface of a unit sphere in 3D space. The new vectors representing the data are the unit vectors from the centre of the sphere to
its surface. The mapping is reversible so no information is lost. We shall call this spherical vector-length normalisation or spherical normalisation for short.

Mapping a plane onto a sphere’s surface may be achieved by many different projections, all of which may be generalised to arbitrary dimensions. One need only look at some of the projections used by cartographers: projections used to make maps of the Earth, of which there are thousands. In our case we are interested in the reverse of these projections but extended to arbitrary dimension. General projections used by cartographers are the azimuthal, conical and cylindrical projections. We consider three different azimuthal projections, which are illustrated in figure 5.3 along with the explicit transformation and the corresponding kernel function. The three projections are the orthographic projection, the stereographic projection and a modified stereographic projection.

Amongst the three, the orthographic projection (figure 5.3a) is possibly the most limited since the input data is restricted to within a small region directly beneath the hemisphere. This mapping may only be used if it can be guaranteed that the data will always lie within that region. The transformation cannot be applied to data that are not situated directly under the hemisphere. This is an undesirable property since, in most problems, the restriction on the data cannot be guaranteed.

The stereographic projection (figure 5.3b) does not suffer from this restriction. However, the space represented by the sphere is wrapped around itself. Consider two points one located near plus infinity and the other near minus infinity in the original space. After applying the stereographic projection the two points will be placed next to each other on the sphere. This means that they will be treated similar to each other. The cyclic property of this space may be useful in applications where all large vectors are treated as similar.

We choose to study the modified stereographic projection (figure 5.3c) in this thesis. It does not suffer from the problems associated with the previous two projections: a point near infinity would be mapped to the equator of the hemisphere and a point near minus infinity would be mapped to an equatorial point directly opposite, on the other side of the hemisphere. The mapping is achieved by augmenting the vec-

---

**Figure 5.2:** Spherical vector-length normalisation: mapping onto a sphere
5.3. Spherically normalised polynomial kernels

\[ \phi(x) = \begin{bmatrix} x \\ \sqrt{1-x^2} \end{bmatrix} \quad \text{for } |x| \leq 1 \]

\[ K(x, y) = x \cdot y + \sqrt{(1-x^2)(1-y^2)} \]

a: The orthographic projection

\[ \phi(x) = \frac{2D}{x^2 + D^2} \begin{bmatrix} x \\ y \end{bmatrix} \]

\[ K(x, y) = \frac{x^2 y^2 - D^2 (x^2 + y^2 - 4 x \cdot y) + D^4}{(x^2 + D^2)(y^2 + D^2)} \]

b: The stereographic projection

\[ \phi(x) = \frac{1}{\sqrt{x^2 + d^2}} \begin{bmatrix} x \\ d \end{bmatrix} \]

\[ K(x, y) = \frac{x \cdot y + d^2}{\sqrt{(x^2 + d^2)(y^2 + d^2)}} \]

c: The modified stereographic projection

Figure 5.3: Spherical vector-length normalisation: various ways of projecting data onto unit hyperspheres.
5.3. Spherically normalised polynomial kernels

tors with a constant, $d$, and normalising the new vector by its Euclidean length. Substituting the length normalised vectors into equation (5.10) we get the spherically normalised polynomial kernel,

$$K_{\text{sphnorm}}(x,y) = \frac{1}{2^n} \left( \frac{x \cdot y + d^2}{\sqrt{(x \cdot x + d^2)(y \cdot y + d^2)} + 1} \right)^n.$$  

The constant, $d$, is the shortest distance from the origin of the hemisphere to the input hyperplane. It determines how the data should be distributed on the sphere. If it is large the data are mapped onto a small region about the pole of the hemisphere. If it is small then the data are mostly mapped onto a strip around the equator: when $d = 0$ then all of the data are mapped onto the equator. The constant is easy to set if the distribution of the data is known. From the geometry of the transformation, if the data have zero mean and unit variance then $d = 1$ maps the data across a large proportion of the hemisphere.

### 5.3.1 RBF interpretation

From equation (5.10), the spherically normalised polynomial kernel may be rewritten,

$$K(x,y) = \left( \frac{\cos \theta_{xy} + 1}{2} \right)^n$$  

since the dot product between two unit vectors, $\hat{x}$ and $\hat{y}$, is equal to the cosine of the angle between, $\theta_{xy}$. Compare it to the RBF kernel,

$$K(x,y) = \exp \left( -\frac{||x - y||^2}{2\sigma^2} \right).$$

Both kernels are constrained in the range $0 \leq K(x,y) \leq 1$. The angle, $\theta_{xy}$, between two vectors to the surface of a unit sphere equals the length of the arc between the two points, represented by the vectors, measured along the surface of the sphere. This distance measure is analogous to the term $||x - y||$ in the RBF kernel. Furthermore, the transformed cosine is a unimodal function where $n$ controls its width. This is similar to the Gaussian where $\sigma$ controls the width. In the RBF kernel, small $\sigma$ gives rise to a narrower RBF and higher data resolution in the input space. In the spherically normalised kernel larger values of $n$ will reduce the half width of the cosine function and give better resolution to smaller angles between vectors. Therefore, the spherically normalised polynomial kernel may be interpreted as a radial basis function kernel.

The interpretation allows us to see why this method should work. Since the kernel is unimodal, ensuring that $K(x,x) > K(x,y) \geq 0$ for $x \neq y$, then the Hessian is diagonally dominant. A diagonally dominant Hessian gives rise to better convergence during optimisation [Lue84].

### 5.3.2 Properties of the spherically normalised general polynomial kernel

From the equation for the general polynomial in table 4.1, a more general form of the spherically normalised polynomial kernel is,

$$K(x,y) = (a\hat{x} \cdot \hat{y} + b)^n$$  

where any real value of $a$ and $b$ are valid and $n$ is an integer greater than zero. However, constraining $a$ and $b$ by the inequalities $a + b \leq 1$ and $b - a \geq -1$ will enforce $-1 \leq a\hat{x} \cdot \hat{y} + b \leq 1$ (since $-1 \leq \hat{x} \cdot \hat{y} \leq 1$) so allowing $n$ to be set arbitrarily large and still prevent the kernel becoming arbitrarily large. These constraints enable very high order polynomial kernels to be used.

Consider $a = 1$, $b = 0$ and $n = 1$ and let the spherical normalisation mapping be the modified stereographic projection. In this case the data exists on the surface of a hemisphere. Imagine a three dimensional.
hemisphere such as the one illustrated in figure 5.2. The types of decision boundaries that may be generated by this kernel may be observed geometrically. Imagine intersecting a plane with the hemisphere in such a way that the plane does not intersect the base of the hemisphere. Bear in mind that, in this projection, the surface of the hemisphere represents the whole of the original input space. The intersection with a plane parallel to the base of the hemisphere near the pole will induce, in the original input space, a small circular decision boundary. Now move the plane closer to the hemisphere’s base. The induced circular decision boundary increases in size. By changing the orientation of the plane, many other closed loop decision boundaries maybe induced on the input space. The induced decision boundary need not be a closed loop. If the plane intersects the hemisphere’s base then the induced decision boundary is open. Indeed, if the plane dissects the hemisphere’s base into two equal halves (when the plane intersects the centre of the base) then a linear decision boundary is induced in the input space.

Let us turn to the effect of the polynomial order, $n$. An unnormalised polynomial kernel SVM induces a decision boundary that has the functional form of a polynomial. That boundary may be described as a polynomial decision surface (or manifold). A spherically normalised polynomial kernel is equivalent to projecting the data onto a hemisphere and then classifying using an unnormalised polynomial kernel SVM. Therefore, in our 3D hemisphere example, the effect of the parameter $n$ is to induce an $n^{th}$ order polynomial decision surface in the 3D space occupied by the hemisphere. The induced decision boundary on the original input space may be obtained from the intersection of the polynomial decision surface with the hemisphere.

The obvious remaining question is, “What is the effect of the parameters $a$ and $b$?” It may be answered, perhaps only partially, by plotting a graph of the kernel,

$$K(x,y) = (acos\theta_{xy} + b)^n. \quad (5.15)$$

as a function of the angle between two vectors, $\theta_{xy}$, $-\pi \leq \theta_{xy} \leq \pi$, for specific values of $a$, $b$ and $n$. Some examples are illustrated in figure 5.4. The immediate effect is to linearly transform the cosine curve. However, subsequently raising to the power $n$ yields an interesting set of kernels. Figure 5.4a illustrates the RBF-like spherically normalised kernel. By varying the parameters it is possible to generate a function mapping that is in a sense opposite to the RBF (figure 5.4b) in that points that are close together in the input space are assigned a low score while points that are far apart (in the input space) and mapped to opposite sides of the hemisphere are assigned a high score. Unfortunately, this particular parameter setting yields a negative definite matrix which means that it is not a valid kernel. A matrix with positive off diagonal elements and zeros along the diagonal is not positive definite. This example shows that not all values of $a$ and $b$ yield valid kernel functions. Figure 5.4c is a combination of figures 5.4a and 5.4b. An alternative interpretation of the right hand graph of figure 5.4c is that it scores the vector pairs (in the space of the hemisphere or sphere) according to how parallel they are (highest scores when $\theta_{xy} = \pm \pi$ or zero). Clearly, further research into this class of kernel function is required, including research into the range of values that $a$ and $b$ may take. Through this interpretation it is possible to assign values to $a$ and $b$ to obtain particular characteristics. Previously, in the general polynomial kernel the same parameters could only be determined by experimentation.

### 5.3.3 Spherical normalisation for any kernel

The ability to apply spherical normalisation to any arbitrary kernel would make this technique powerful, especially as it would impart upon any valid kernel properties similar to the RBF kernel as described above. The expression for spherically normalised kernel shown in figure 5.3c is entirely in terms of dot products
5.3. Spherically normalised polynomial kernels

\[ K_{xy}^{-p} \]

- **a:** Spherical normalisation with an RBF interpretation is obtained by setting the kernel parameters \( a = 0.5 \) and \( b = 0.5 \). The parameter \( n \) controls the width: \( n = 1 \) (left), \( n = 3 \) (centre), \( n = 6 \) (right). Vectors that make a small angle between them are given a high score. Vectors with a large angle are given a low score.

\[ K_{xy}^{-p} \]

- **b:** An unusual anti-RBF interpretation that is the opposite of (a) is obtained by setting the kernel parameters \( a = -0.5 \) and \( b = 0.5 \). The parameter \( n \) controls the width: \( n = 1 \) (left), \( n = 3 \) (centre), \( n = 6 \) (right). In this case vectors that make large angles are given a high score while vectors that make small angles have a low score. Note that a matrix with diagonal zero elements and positive off diagonal elements is not positive definite so this parameter setting does not give a valid kernel. This shows that parameters of the general polynomial must be chosen with care.

\[ K_{xy}^{-p} \]

- **c:** A highly unusual RBF that combines both (a) and (b) interpretations is obtained by setting the kernel parameters \( a = 1 \) and \( b = 0 \). The parameter \( n \) that controls the widths, \( n = 1 \) (left), \( n = 3 \) (centre), \( n = 6 \) (right), also controls the relative direction (whether positive or negative) of influence of the RBF interpretations from (a) and (b) depending upon whether \( n \) is even or odd.

Figure 5.4: Plots of the spherically normalised general polynomial kernel output scores as a function of the angle between vectors illustrate the effect of choosing different values of the kernel parameters.
Table 5.3: Error rates of spherically normalised polynomial kernels trained on YOHO

<table>
<thead>
<tr>
<th>Classifier type</th>
<th>poly. order</th>
<th>test on training set impostors</th>
<th>test on unseen impostors</th>
<th>support vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>polynomial</td>
<td>2</td>
<td>1.30</td>
<td>1.65</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>0.18</td>
<td>0.38</td>
<td>1</td>
</tr>
<tr>
<td>SVM</td>
<td>2</td>
<td>2.13</td>
<td>2.50</td>
<td>742</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.74</td>
<td>1.05</td>
<td>725</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.51</td>
<td>0.72</td>
<td>699</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>0.41</td>
<td>0.53</td>
<td>722</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0.34</td>
<td>0.46</td>
<td>791</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>0.32</td>
<td>0.43</td>
<td>837</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>0.30</td>
<td>0.41</td>
<td>793</td>
</tr>
</tbody>
</table>

between vectors. By substituting the dot products with another kernel function, we obtain the equation for a general method of normalising any kernel that yields an ill-conditioned Hessian.

\[ K_{\text{sphnorm}}(x, y) = \frac{K(x, y) + d^2}{\sqrt{(K(x, x) + d^2)(K(y, y) + d^2)}} \]  

Equation (5.16) induces the mapping onto the sphere only. To introduce the RBF interpretation (and other properties described earlier) we must include the parameters of the general polynomial kernel, thus,

\[ K_{\text{sphnorm}}(x, y) = \left( a \frac{K(x, y) + d^2}{\sqrt{(K(x, x) + d^2)(K(y, y) + d^2)}} + b \right)^n \]  

In these cases the spherical normalisation occurs in the feature space induced by the first kernel, \( K \). This form of spherical normalisation may be used to normalise a kernel for which an explicit form of the input to feature space transformation is not known. These equations have been applied in some preliminary work involving pair HMMs (section 7.3).

It is simple to verify that the spherically normalised kernels are valid kernels to use with SVMs. The criterion for a valid kernel is that it must represent the dot product between two vectors in some other space,

\[ K(x, y) = \Phi(x) \cdot \Phi(y) \]  

where \( \Phi \) is a transformation to some feature space, which can equally be expressed as two transformations applied one after another:

\[ K(x, y) = \phi_2(\phi_1(x)) \cdot \phi_2(\phi_1(y)). \]  

The spherically normalised polynomial kernel combines two transformations. The first transformation, \( \phi_1 \), is the mapping of the original kernel, \( K \), and the second, \( \phi_2 \), is the transformation onto a hypersphere. Thus the spherically normalised kernel represents a dot product.

### 5.3.4 Experiments

Spherically normalised polynomial kernel SVMs were trained on YOHO using data normalised to zero mean and unit variance and quantised to 100 codebook vectors. The regularisation and spherical normalisation parameters of the SVMs were set to one. Test results are shown in table 5.3. The spherical
normalisation improved the accuracy of the second order SVM from 3.37% to 2.50% EER. For the same order polynomial, the SVM is not as accurate as the polynomial classifier but this may be an effect of vector quantisation. Spherical normalisation also enabled higher order SVMs to be trained. Despite the information lost through vector quantisation, the 14th order SVM is capable of matching the test accuracy of the polynomial classifier. This is an encouraging result for the SVM.

5.4 Score-space kernels

The score-space[SGN01] approach described in section 4.5 enables SVMs to classify variable length sequences. This is done by combining a generative model (in our case a GMM) with the SVM via the kernel function. This approach is significantly different from the frame-based strategies adopted in the previous two sections, where the discriminative classifier scored each frame of an utterance directly and the utterance score was defined as the mean frame score. In that approach the discrimination by the classifier is at the frame-level only. By definition a discriminative approach discards information that is seen as irrelevant. Therefore, the SVMs discard all information that is not relevant to scoring each frame. Information that is important for discriminating between sequences may be discarded inadvertently. In speaker verification the utterance score is desired. A frame-level discriminative approach is, therefore, not ideal for scoring utterances. In this section we use the score-space kernel as method of scoring utterances in a time-independent fashion (the utterance score remains independent of the ordering of the frames) without scoring individual frames.

The score-space approach is better in that it discriminates at the utterance-level. Instead of classifying one frame at a time, this approach allows the SVM to score whole utterances directly by treating complete sequences as single input vectors. Using a generative model, a modelling strategy that is designed to capture all of the variations in the data, to map a sequence to a fixed length vector should hopefully retain more of the information that is useful for utterance-level discrimination. By performing the discriminative scoring at the last possible moment less information is lost and the discriminative classifier is able to assign a score more accurately.

Further benefits of the score-space approach is that it reduces the number of training vectors observed directly by the SVM. Instead of observing every frame of every training utterance it need only see one vector per utterance. This reduction simplifies SVM training and removes the need for vector quantisation, which was just a way of reducing the number of support vectors.

The support vectors of an SVM trained using these kernels represent the set of borderline utterances, or support utterances, that define the decision boundary. This has an analogy with many speaker recognition systems that use distance measures to choose a subset of speakers from the pool of background speakers to train against each client. For example, the method by Reynolds [Rey95] used a log likelihood distance measure, firstly, to select the background speakers who were most similar to the client and, secondly, to select a set of speakers that would uniformly represent the rest of the world. Two GMMs would be trained, one on the client’s speech and the other on the subset of background speakers. The discriminative SVM approach combines the background speaker selection and classifier training processes into a single optimisation procedure. The support vector machine keeps only the data that define the decision boundary. Background speakers (and this also applies to the client’s data) that are easily classified are discarded leaving only the support vectors. The subset of the speakers defined by the support utterances are those who are most similar to the client.
Table 5.4: GMM baseline performance on YOHO.

The classifier labelled * was trained and tested under different conditions. The result is quoted from [Rey95].

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Average EER %</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMM</td>
<td>1.08</td>
</tr>
<tr>
<td>GMM-LR *</td>
<td>0.58</td>
</tr>
<tr>
<td>GMM-LR</td>
<td>0.00</td>
</tr>
</tbody>
</table>

5.4.1 The baseline GMM

The most appropriate baseline classifier for the score-space SVM is the GMM that is used to derive the score-space kernel mapping. The baseline client GMMs (one GMM per client) had 512 diagonal covariance Gaussians. For 24 dimensional data there is a total of 25088 adjustable parameters per model. Since the speakers were partitioned into two groups then two world GMMs were trained, one for each of the groups. Each world GMM, consisting of 512 diagonal covariance Gaussians, was trained using the data from all of the speakers in its group. In an experiment involving the likelihood ratio of the client model to the world model, a client model in group one would be combined with the world model of group one. Thus the speakers of group two may be used as unseen impostors in a fair evaluation. As a consequence, there exists a small overlap in the training data of the client and world models. Fortunately, since the world models are trained on all 69 speakers of the respective group then they are significantly better world models than they are models of any one client.

The baseline results are shown in table 5.4. Notice that the likelihood ratio GMM (GMM-LR) with the world model consisting of 69 speakers achieves zero error on this database. The GMM-LR result reported by Reynolds [Rey95] does not achieve this level of performance since significantly fewer speakers were used to build the world models. We shall, never-the-less, continue to develop our techniques on the YOHO database but the benefits of any method involving the likelihood ratio of two Gaussian mixture models cannot be assessed on this database.

5.4.2 Experiments

The underlying generative model used to derive the score-space is the baseline GMM described above. To recap one GMM consisting of 512 components with diagonal covariance matrices was trained for each speaker. With 24 dimensional input data this corresponds to models with 25088 parameters. The Fisher kernel, therefore, maps to a 25088 dimensional space and the Fisher kernel with argument maps to a 25089 dimensional space. To put it another way, the variable length utterances are all mapped to a fixed length vector of that size. For the YOHO database the utterance length is on average about 200 frames long. Each frame is a 24 dimensional vector, so the kernel is typically mapping from a 4800 dimensional space — an increase by about a factor of 5 in the dimensionality. This is an ideal task for the SVM. Unregularised classifiers cannot be trained reliably on 25088 dimensional score-space vectors using only 6555 training examples without over-training. Note that the likelihood ratio kernel has not be tested on YOHO since the likelihood ratio GMM is already achieving zero error — any benefit due to the kernel will go unnoticed. The Fisher kernel matrices were computed using equations (4.41) to (4.44).
Table 5.5: Performance of Fisher kernel SVMs on text independent speaker verification and speaker identification tasks.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Average EER %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline GMM</td>
<td>1.08</td>
</tr>
<tr>
<td>Fisher kernel - no score-space whitening</td>
<td>0.68</td>
</tr>
<tr>
<td>Fisher kernel - score-space whitened</td>
<td>0.30</td>
</tr>
<tr>
<td>Fisher kernel + argument - score-space whitened</td>
<td>0.27</td>
</tr>
</tbody>
</table>

Table 5.6: Detailed breakdown of Fisher kernels on text independent speaker verification

<table>
<thead>
<tr>
<th>Fisher kernel</th>
<th>Average EERs % (seen)</th>
<th>Average EERs % (unseen)</th>
<th>Number SVs</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>without whitening</td>
<td>0.52</td>
<td>0.68</td>
<td>155</td>
<td>28</td>
</tr>
<tr>
<td>with whitening</td>
<td>0.19</td>
<td>0.30</td>
<td>387</td>
<td>23</td>
</tr>
<tr>
<td>with argument &amp; whitening</td>
<td>0.16</td>
<td>0.28</td>
<td>386</td>
<td>23</td>
</tr>
</tbody>
</table>

Since the inverse of a full covariance matrix of 25088 dimensional vectors is impractical to compute it was necessary to assume that the principle components of score-space are already orthogonal. The score-space vectors were normalised to zero mean and unit diagonal covariance, a process called whitening. To illustrate the necessity of whitening, a set of Fisher kernel SVMs were trained without normalising to zero mean and unit diagonal covariance. Finally, to investigate whether the likelihood score contains any more information than the derivatives of the model (and vice-versa) a set of likelihood score-space with argument kernel SVMs were trained. Results are shown in table 5.5 and a more detailed break-down in table 5.6.

The results show clearly that score-space whitening is required — even if it is just a poor approximation by rescaling the score-space components to zero mean and unit variance. By the results of lines 1 and 3 in table 5.5 it is clear that the Fisher kernel is yielding additional information that is not already encoded in the likelihood score. From line 4 where the log likelihood score is also included as an additional dimension in the score-space there is a small improvement in the performance over line 3. This tends to suggest that the log likelihood score does encode information, albeit a small amount, that is not already contained in the derivatives of the parameters. This hypothesis shall be verified in the next chapter.

The results from the YOHO experiments are quite promising but none approach the performance of the GMM likelihood ratio. It is yet to be seen whether the likelihood ratio score-space kernel will outperform the GMM likelihood ratio. This will the subject of the next chapter where all of the methods examined so far will be re-evaluated on PolyVar, a more difficult task.

# 5.5 Conclusion

In this chapter we compared the properties of a polynomial kernel SVM with a polynomial classifier and identified a number of issues that affect the SVM performance adversely. Firstly, the data must be normalised (whitened). Ideally, normalisation should take place in the feature space where the margin is maximised. When that is not possible then input space normalisation will suffice (for the polynomial kernel).
Secondly, the numerical magnitudes of the kernel function influences the ability of the optimiser to find a solution. The spherical vector-length normalisation technique solved this problem by preconditioning the Hessian.

Spherical normalisation is highly significant to SVMs. It has many beneficial properties:

1. It preconditions the Hessian making the quadratic programming optimisation of SVMs easier.
2. It rescales, for example, polynomial kernels by preventing them yielding large values.
3. It has an RBF interpretation in the mathematical formulation of the kernel.
4. The effect of the parameters of the general polynomial kernel are more easily understood.
5. It may be applied on top of any valid kernel and impart upon it properties of RBFs.

The method of spherical normalisation has yielded a new family of kernels. Different properties may be imparted on the spherical normalisation kernel depending upon the specific projection used to map to a sphere and upon the parameters chosen for the kernel.

Also in this chapter, we obtained some initial results on applying the likelihood score-space (Fisher) kernel to text independent speaker verification that outperformed the underlying baseline GMM from which the kernel is derived. The result shows that the Fisher kernel is an effective way of performing sequence-level time independent variable length sequence classification.

### 5.5.1 The polynomial sequence kernel

Since this work was done, the sequence polynomial kernel has been developed by Campbell [Cam02], which also removes the dependence upon vector quantisation. Let us illustrate the principle here. By computing the polynomial expansion explicitly it is possible to take advantage of the linearity of the SVM to reduce the quantity of training data. The utterance score is defined by,

\[
S(X) = \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{s} y_s \alpha_s \phi(x_s) \cdot \phi(x_i) + b \right)
\]

\[
= \sum_{s} y_s \alpha_s \phi(x_s) \cdot \left( \frac{1}{N} \sum_{i=1}^{N} \phi(x_i) \right) + b \quad (5.20)
\]

where \(s\) enumerates the support vectors, \(\phi(x)\) is the explicit polynomial expansion of a frame in the utterance \(X = \{x_1, \ldots, x_N\}\). Since the polynomial expansion is computed explicitly then their means may be computed and used to represent the complete sequence \(X\). Campbell used this representation to train SVMs for speaker verification [Cam02].
Chapter 6

Evaluation on PolyVar

6.1 Introduction

When developing new techniques to solve a given problem it is possible to devise new methods that perform well on that task. However, there is always a chance that those methods may fail on a different task. When new methods fail to be ported to another task then it is likely that those methods have exploited some features of the original problem that do not exist in the new one. This is analogous to overtraining a classifier to model noise in data but on a much higher level. One of the aims of this chapter is to determine whether the techniques developed on YOHO, spherical normalisation in particular, may be applied more generally across text independent speaker verification. To achieve this aim it is necessary to re-run the simulations of the previous chapter on a new database: the PolyVar database described in section 2.4.2.

The YOHO database has been traditionally used by researchers to report speaker recognition results. Compared to other databases, the speech signals in YOHO are clean and their overall quality is high. Error rates are correspondingly low. There are disadvantages to this. When data is clean, a classifier may perform particularly well. However, the same performance may not carry over to noisy data. There is concern that this may be true for SVMs. Recall that the solution of an SVM contains all the training data that the SVM misclassifies. When the data is noisy then the classifier will have a higher error rate and the SVM solutions will increase in size. The SVM is at an immediate disadvantage compared to parametric classifiers as the solutions will be larger. PolyVar is a more difficult task. The speech signals contain more noise so error rates are higher compared to those on YOHO. The second aim of this chapter is to determine what effect noisy data has on the SVMs. We show that the effect of using noisy data on the performance of SVMs is not much more adverse than on the performance of the GMM baselines.

Due to the cleanliness of the data and the small number of client test utterances, error rates on YOHO are close to zero. Also the small number of client test compared to the number of impostor tests places a shadow over the reliability of the YOHO results. This made results difficult to analyse and interpret. It has already been demonstrated that a well trained GMM system can achieve zero error on YOHO. As a direct consequence it was not possible to evaluate the likelihood ratio score-space kernel SVM. It is, therefore, difficult to make any definitive conclusions. A more difficult database such as PolyVar, which also has a much greater number of client tests, will enable more concrete conclusions to be made. The third aim is to perform an evaluation of the likelihood ratio score-space kernel SVM and finally to validate the conclusions drawn from the previous chapter. We show that likelihood ratio score-space kernel SVM outperforms the GMM likelihood ratio baseline. We also show that spherically normalising the likelihood ratio score-space
kernel yields a further, more significant drop in the error rates.

Some of the results in this chapter have been published in [WR03]. A description of the baseline classifiers and how they were trained is given in section 6.2. They include the GMM and GMM-LR classifiers. Additionally there is a third classifier which gives the state-of-the-art result on PolyVar. This is the GMM-LR/SVM approach which was reviewed in section 3.4.2. There is no polynomial classifier baseline for this evaluation. Section 6.3 describes the experiments and results of the evaluation using polynomial SVMs (section 6.3.1) and score space kernels (section 6.3.2). Firstly, a description of the experimental procedure is necessary.

6.1.1 PolyVar protocol for training and testing

Details of the PolyVar database can be found in section 2.4.2. The audio was processed using perceptual linear prediction with a 32ms window and a 10ms time shift to obtain 12 cepstral coefficients and one energy term. These features were augmented by their first and second order derivatives so that each frame consisted of a 39 dimensional vector. The data was normalised so that the distribution of frames from all speakers had zero mean and unit diagonal covariance.

The silence vectors were stripped off the beginnings and ends of the utterances using the information contained in the headers of each file on the CDROM. Silence vectors within each utterance were segmented out using an MLP trained on spoken English. Although there is a mismatch in the phone set the neural network could still detect silence vectors accurately.

Included on the PolyVar CDROM is a strict training and testing protocol that lists the client training utterances, a set of impostor utterances and for each client, a set of test utterances that includes utterances from unseen impostors. This protocol was adhered to strictly. For each of the 38 speakers chosen as clients there where 5 utterances of 17 different phrases making 85 training utterances per client. There were 952 utterances for training the world model, which consisted of 56 different utterances of each of the 17 phrases. For testing, each client between 350 and 400 test utterances and were tested against approximately 600 unseen impostor utterances.

6.2 The GMM baselines

The GMM and GMM likelihood ratio (GMM-LR) classifiers described in section 3.2.2 are used as the baseline classifiers and as the generative models for the score-space mappings. The classifier that gives state-of-the-art results is the combined GMM-LR/SVM system reviewed in section 3.4.2. Results on the PolyVar database for the GMM-LR and the GMM-LR/SVM have already been published by Bengio and Mariéthoz [BM01]. To make a fair evaluation the baseline models were trained with exactly the same parameters used by Bengio and Mariéthoz.

The world GMM contained 1000 Gaussians. This number was obtained by Bengio and Mariéthoz using cross-validation. Models of various sizes were trained using 90% of the training data while the remaining 10% was used as cross-validation data to test the models. The model with the highest likelihood on the cross-validation data was chosen.

The number of Gaussians in the client model was found using ten-fold cross-validation. The training data was randomly partitioned into ten equally sized sets. Every client model was trained on nine of the sets and tested on the one remaining set. This was repeated ten times, each time with a different test set. Repeating the whole process using different numbers of Gaussians in the models Bengio and Mariéthoz found the model that yielded the highest overall likelihood had 200 components.
6.3 SVM evaluations

Using the GMM-LR, Bengio and Mariéthoz reported a text independent speaker verification result of 5.55% HTER on the PolyVar test data using 19 speakers. A 4.73% HTER was reported for the GMM-LR/SVM. The results obtained using 38 speakers are shown in table 6.1 and their corresponding DET curves have been plotted in figure 6.1.

It is interesting to compare the result of the baseline GMM-LR (6.12% EER) with the GMM-LR/SVM (5.94% EER). While the equal error rates suggest that the GMM-LR/SVM system is better the DET curve shows an interesting trend in the operating characteristics. At very low false acceptance and very low false rejection probabilities it is actually the GMM-LR system that performs better. Only near the diagonal of the graph (corresponding to the equal error rate) does the GMM-LR/SVM system perform better than the GMM-LR. The GMM-LR/SVM is a highly tuned classifier that trades-off the accuracy at the tails of the DET curve to gain a small increase in accuracy near to the diagonal. The improvement in the GMM-LR/SVM result has arisen by changing the relative weights of the client model and the world model when scoring the utterances (see equations in section 3.4.2). The SVM creates a bias towards either the client or world model in order to minimise the error on the training data, but no new information is added and the utterance score is no longer Bayes optimal. Therefore, other operating points on the DET curve will have become less optimal. However, the strength of the GMM-LR/SVM method should not be overlooked: if a particular false acceptance to false rejection ratio is desired it may be possible to use a similar method to achieve a slightly better performance than the basic GMM-LR for that particular operating ratio.

6.3 SVM evaluations

Experiments performed chapter 5 were individually selected to be repeated using the new database. All of the results are summarised in table 6.1. Figures 6.2 to 6.5 show the DET profiles that compares the relative performance of the different SVM approaches to the baselines. Specific details and an analysis of each evaluation follows.

6.3.1 Polynomial kernels

Some of the polynomial experiments performed on YOHO were repeated here to verify the conclusions drawn in the previous chapter. The same methods applied in chapter 5 to train polynomial and normalised polynomial kernels are used here. Vector quantisation was done on the training data to reduce its quantity and limit the size of the SVMs. 1500 codebook vectors were computed for both clients and impostors using nearest neighbour clustering. The training data for each SVM, therefore, consists of 3000 codebook vectors. The SVMs were trained using a linear loss function and a regularisation parameter, C = 1. In theory, the regularisation parameter is only important for second order polynomials. For 39 dimensional input vectors the second order polynomial kernel maps to a feature space of 820 dimensions and the third order polynomial kernel maps to an 11480 dimensional space. Once the feature space dimensionality exceeds the number of training vectors then the classification problem becomes linearly separable and the SVM formulation for linear separable problems (that does not need C) may be used. Never-the-less, C = 1 was used for all polynomial kernel experiments.

The results of training various orders of polynomial and normalised polynomial kernels are shown in table 6.1. The second and third order unnormalised polynomial kernels were trained. The results were poor but this was expected. The significant drop in performance in the third order kernel, which is almost random, compared to the second order kernel was again due to the ill-conditioned Hessian that results from
6.3. SVM evaluations

Table 6.1: Results of the PolyVar experiments

The GMM baseline consists of 200 diagonal covariance Gaussian components modelling the clients only. The GMM likelihood ratio (GMM-LR) baseline consists of the above plus a GMM with 1000 diagonal covariance Gaussian components modelling the impostors. The GMM-LR/SVM is the state-of-the-art classifier, which uses the same client and impostor GMMs but computes a weighted log likelihood ratio. The spherical normalisation constant was set to one in all instances where it was used. The half total error rates (HTER) and equal error rates (EER) are shown along with the standard deviation in brackets.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>% HTER (s.d.)</th>
<th>% EER (s.d.)</th>
<th>Avg. support vectors (s.d.)</th>
<th>Avg. no. parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GMM</td>
<td>11.22 (7.19)</td>
<td>12.07 (7.76)</td>
<td>—</td>
<td>15800</td>
</tr>
<tr>
<td>GMM-LR</td>
<td>5.53 (6.14)</td>
<td>6.12 (6.96)</td>
<td>—</td>
<td>94800</td>
</tr>
<tr>
<td>GMM-LR/SVM</td>
<td>5.37 (6.26)</td>
<td>5.94 (6.83)</td>
<td>—</td>
<td>94803</td>
</tr>
<tr>
<td><strong>Polynomial kernel</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>order 2</td>
<td>15.53 (8.62)</td>
<td>16.49 (8.96)</td>
<td>255 (27)</td>
<td>10200</td>
</tr>
<tr>
<td>order 3</td>
<td>45.82 (4.59)</td>
<td>51.82 (9.42)</td>
<td>105 (30)</td>
<td>4200</td>
</tr>
<tr>
<td><strong>Normalised polynomial kernel</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>order 2</td>
<td>14.47 (8.13)</td>
<td>15.51 (8.49)</td>
<td>802 (166)</td>
<td>32080</td>
</tr>
<tr>
<td>order 3</td>
<td>10.22 (7.20)</td>
<td>10.94 (7.52)</td>
<td>884 (109)</td>
<td>33760</td>
</tr>
<tr>
<td>order 5</td>
<td>10.20 (7.59)</td>
<td>10.85 (7.88)</td>
<td>2072 (202)</td>
<td>82880</td>
</tr>
<tr>
<td><strong>Fisher kernel</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without argument</td>
<td>6.54 (5.01)</td>
<td>6.98 (5.24)</td>
<td>396 (29)</td>
<td>63201</td>
</tr>
<tr>
<td>without argument &amp; sph. norm</td>
<td>6.50 (4.29)</td>
<td>6.87 (4.22)</td>
<td>677 (35)</td>
<td>63201</td>
</tr>
<tr>
<td>with argument</td>
<td>6.50 (5.05)</td>
<td>6.92 (5.23)</td>
<td>394 (29)</td>
<td>63204</td>
</tr>
<tr>
<td>with argument &amp; sph. norm</td>
<td>6.47 (4.22)</td>
<td>6.87 (4.27)</td>
<td>675 (34)</td>
<td>63205</td>
</tr>
<tr>
<td><strong>LR kernel</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>without argument</td>
<td>5.13 (5.19)</td>
<td>5.55 (5.62)</td>
<td>143 (23)</td>
<td>379201</td>
</tr>
<tr>
<td>without argument &amp; sph. norm</td>
<td>3.72 (4.41)</td>
<td>4.03 (4.64)</td>
<td>946 (23)</td>
<td>379201</td>
</tr>
<tr>
<td>with argument</td>
<td>5.03 (5.12)</td>
<td>5.55 (5.68)</td>
<td>143 (23)</td>
<td>379204</td>
</tr>
<tr>
<td>with argument &amp; sph. norm</td>
<td>3.71 (4.41)</td>
<td>4.03 (4.67)</td>
<td>944 (23)</td>
<td>379204</td>
</tr>
</tbody>
</table>
The DET curves of some of the baseline classifiers is shown: the Gaussian mixture model (gmm), the Gaussian mixture model likelihood ratio (gmm-lr) and the combined GMM-LR/SVM classifier (gmm-lr/svm) by Bengio and Mariethoz.

Figure 6.1: DETs of the baseline classifiers on PolyVar
6.3. SVM evaluations

The DET curve of the polynomial kernel SVM simulations are shown: the second order polynomial kernel (poly, 2), the spherically normalised second order polynomial kernel (sp norm poly, 2), the spherically normalised third order polynomial kernel (sp norm poly, 3), the spherically normalised fifth order polynomial kernel (sp norm poly, 5). For reference, the baseline GMM (gmm) and GMM-LR (gmm-lr) systems are shown.

Figure 6.2: DETs of the polynomial kernel SVMs on PolyVar
The DET curve of the Fisher kernel simulations: the fisher kernel SVM (fisher kernel) is shown along side the GMM (gmm) and GMM-LR (gmm-lr) baselines.

Figure 6.3: DETs of the Fisher kernel SVMs on PolyVar
The DET curve of the likelihood ratio score-space kernel simulations these are: the likelihood ratio score-space kernel that is not spherically normalised (basic lr kernel) and the likelihood ratio score-space kernel with spherical normalisation (sp norm lr kernel). The baseline GMM-LR profile (gmm-lr) is included for reference.

Figure 6.4: DETs of the likelihood ratio score-space kernel SVMs on PolyVar
6.3. SVM evaluations

The DET curve of some of the classifiers is shown: a fifth order spherically normalised polynomial kernel SVM (normalised poly, 5), the baseline Gaussian mixture model (gmm), the baseline likelihood ratio GMM (gmm-lr), the likelihood score-space kernel SVM without spherical normalisation (fisher kernel), the likelihood ratio score-space kernel SVM with spherical normalisation (lr kernel) and the previous state-of-the-art combined GMM-LR/SVM classifier (gmm-lr/svm) by Bengio and Mariethoz.

Figure 6.5: Comparison of all approaches on PolyVar
Table 6.2: Effect of vector quantisation on classification performance of a fifth order spherically normalised polynomial kernel SVM. The quantisation of the background speakers was kept constant at 1500 codebook vectors while the number of client codebook vectors was varied.

<table>
<thead>
<tr>
<th>Client VQ centres</th>
<th>% HTER (s.d.)</th>
<th>% EER (s.d.)</th>
<th>Number of SVs (s.d.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>13.53 (8.07)</td>
<td>14.20 (8.25)</td>
<td>686 (43)</td>
</tr>
<tr>
<td>600</td>
<td>12.56 (7.94)</td>
<td>13.22 (8.17)</td>
<td>962 (72)</td>
</tr>
<tr>
<td>800</td>
<td>11.44 (7.71)</td>
<td>12.20 (7.93)</td>
<td>1238 (98)</td>
</tr>
<tr>
<td>1000</td>
<td>10.86 (7.64)</td>
<td>11.44 (7.84)</td>
<td>1493 (126)</td>
</tr>
<tr>
<td>1200</td>
<td>10.32 (7.58)</td>
<td>11.01 (7.92)</td>
<td>1730 (152)</td>
</tr>
<tr>
<td>1500</td>
<td>10.20 (7.59)</td>
<td>10.85 (7.88)</td>
<td>2072 (202)</td>
</tr>
</tbody>
</table>

the unnormalised polynomial. The unnormalised fourth order polynomial kernel could not be trained: the optimiser failed to converge to any solution.

Spherical normalisation identical to that used in the YOHO experiments was applied here,

\[ \mathbf{x} \rightarrow \phi(\mathbf{x}) = \frac{1}{\sqrt{\mathbf{x}^2 + d^2}} \begin{bmatrix} \mathbf{x} \\ d \end{bmatrix} \] (6.1)

The spherical normalisation parameter, \( d \), was set to one since the distribution of the input data was normalised to zero mean and unit variance. Spherically normalising the second order polynomial yielded an improvement in performance from 16.49% EER to 15.51% EER. Normalising the third order polynomial kernel enabled the optimiser to achieve a more reasonable solution, a solution that is significantly better than the second order kernel. Spherical normalisation enables higher order polynomial kernels to be trained. The fifth order normalised polynomial kernel improves upon the third order normalised polynomial kernel, albeit by a small amount.

It is unlikely that training on successively higher order polynomial kernels will achieve a performance close to the GMM-LR and GMM-LR/SVM baselines. An investigation into the effect of the number of vector quantisation codebook vectors on the performance of the SVMs was done using a fifth order normalised polynomial kernel. While keeping the number of codebook vectors for the background speakers fixed at 1500 the amount of quantisation on the client was varied. The results are shown in table 6.2. It can be seen that reducing the number of codebook vectors by a few hundred (by 300 from 1500 to 1200) is more detrimental on the test performance than decreasing the polynomial order from 5 to 3. The key limiting factor of this approach, as it was on YOHO, is the reliance upon vector quantisation to limit the amount of computation required during training and testing. PolyVar is a more noisy database than YOHO. More complex decision boundaries created from higher order polynomial kernels are unlikely to increase the performance without additional training data since the extra modelling capacity of the decision boundary will, most likely, model the noise in the training data. It is unfortunate that SVMs are not well suited to large quantities of training data.

From the DET curves shown in figure 6.2 the polynomial kernel SVMs clearly have a steeper gradient than the baselines. This indicates that the variance of the client utterance scores is smaller than the variance of the impostor utterance scores implying that the output scores of the SVM are more consistent for the client and more varied for the impostors. As a result of the difference in gradient the polynomial SVMs perform better than the baselines at low false rejection probabilities. Compared to the GMM the fifth order spherically normalised polynomial kernel SVM performs better at false rejection probabilities less than
about 22%. Below 0.5% false rejection probability it is better than the GMM-LR baseline also. However, vector quantisation has a significant effect upon this result. Increasing the number of codebook vectors is likely to improve the performance of the SVM further.

From these results we can say that the normalised polynomial kernel SVM, under the right conditions, is more accurate than the baseline systems. In terms of overall performance the SVMs are potentially more accurate than the GMM but less accurate than the GMM-LR. This classifier ranking matches that obtained on YOHO. The results also confirm the effectiveness of spherical normalisation as a fix for the scaling problem in the polynomial kernel (in which high-order polynomials result in ill-conditioned optimisation problems).

### 6.3.2 Score-space kernels

The number of training utterances per SVM for the Fisher and LR kernels was 85 client utterances and 952 background utterances making a total training size of 1037 utterances per SVM. Using a score-space kernel, each utterance is mapped onto a single score-vector. Therefore, each SVM is trained on 1037 vectors. Solving the optimisation problem for a training set of this size is straightforward and does not require any of the iterative chunking methods outlined in section 4.6 that have been developed to train SVMs on large quantities of data.

The Fisher kernel uses the derivatives of the client GMMs of the baseline systems to achieve the score-space mapping. The likelihood ratio kernel uses both client and world models of the baseline GMM-LR system. The number of parameters in the GMM and GMM-LR baselines are 15800 and 94800 respectively. The score-space dimensionality of the Fisher and LR kernels are thus 15800 and 94800 respectively. Using the corresponding score-operator that includes the argument, the score-space dimensionalities become 15801 and 94801 respectively since they include an additional component which corresponds to the output score of the baseline GMMs. In theory, by including the baseline system’s output score, the SVM should have a minimum performance that equals that of the baseline system. The 1037 training score-vectors are normalised (or whitened) so that they are distributed with zero mean and unit diagonal covariance.

The high score-space dimensionalities, particularly that of the LR kernel, caused computational problems for SVMs. The first problem is the memory required to compute the Hessian and Gram matrices of the SVM. To compute these matrices efficiently the training data must be held in memory. When computing the Hessian, the memory needed to store the training score-vectors is 94800 × 1037 × 8 bytes or 750 MB, assuming double precision floating point arithmetic is used. The Gram matrix may require even more memory depending upon the number of support vectors in the solution. There are just under 1000 test utterances per client. During testing, the worse possible case, in which the entire training set are support vectors, would require 94800 × (1037 + 1000) × 8 or 1.43 GB or memory (although the number of support vectors may be reduced to one in this case — this is discussed below). To reduce the memory requirements, the SVM optimisation matrices were computed in small blocks leaving score-vectors that are not in use stored on hard disk. This allows us to use the complete score-vector derived from the GMMs.

The second problem that may arise from the high score-space dimensionality, depends upon the numerical values of the data. It is similar to the problem encountered in chapter 5. In this case the problem may be caused by computing dot products between high dimensional vectors. Depending on the actual numerical values, the dot product between two high dimensional vectors may be very large, but it could also be very small. As explained in chapter 5 this leads to an optimisation problem that is more difficult to solve due to an ill-conditioned Hessian. The solution is to apply the spherical normalisation technique that was developed in section 5.3. In experiments where spherical normalisation is used the mapping onto
the hypersphere (figure 5.3c) is applied explicitly to the score-vectors for computational efficiency. The spherical normalisation transformation is,

\[
\Psi(X) := \phi(\Psi(X)) = \frac{1}{\sqrt{\Psi(X) \cdot \Psi(X) + d^2}} \begin{bmatrix} \Psi(X) \\ d \end{bmatrix}
\]

(6.2)

where \(\Psi(X)\) is the score-vector of the utterance, \(X\). It is not necessary to use the kernel function form of the normalisation. Since the components of the score-vectors are normalised to zero mean and unit variance a suitable value to use for the spherical normalisation parameter, \(d\), is one.

SVMs were trained using the following kernels: Fisher kernel, Fisher kernel with argument, LR kernel and LR kernel with argument. For each kernel, SVMs were trained with and without spherical normalisation. These SVMs are linear classifiers in the score-space. It is possible to use a static kernel on these SVMs, such as the polynomial, higher order spherically normalised polynomial or RBF kernels, to make non-linear decision boundaries in score-space. However, given that the dimensionality of the score-space is significantly higher than the number of training vectors then the classification problem is linearly separable and it is unnecessary to use non-linear boundaries. Also, since the problem is known to be linearly separable the regularisation parameter was set to infinity (i.e. the formulation for an SVM that maximises the margin when the data is linearly separable was used). To give a indication of the value of regularisation parameter (the upperbound on the Lagrange multipliers) that should be used if more regularisation was needed, the Lagrange multipliers in the spherically normalised likelihood ratio score-space kernel SVMs had a mean about 0.25 and an average maximum value of about 4.

The GMM baseline from which the Fisher kernel is derived yielded 12.07% average EER. The Fisher kernel almost halves the average EER to about 6.9%. This is a highly significant reduction and is in line with the result obtained from the YOHO experiments in which a similarly large drop in the EER was seen — but not so large that it outperforms the baseline GMM-LR. The application of spherical normalisation yielded a small reduction in the equal error rate comparable in magnitude to including the argument to the score-operator. The reduction is so small that it is most likely statistically insignificant. Spherical normalisation has little effect in this case because the Hessian of the SVMs are not ill-conditioned. The significance of the result is more clearly seen on the DET profiles shown in figure 6.3. The four variations of the Fisher kernel (shown in table 6.1) are represented as a single curve since they are almost indistinguishable from each other. The curve shown is that of the non-spherically normalised Fisher kernel without argument. It is clear that the Fisher kernel SVM has improved upon the GMM at all points on the DET such that the profile is very close to the GMM-LR baseline.

The results of the non-spherically normalised likelihood ratio score space kernel with and without argument SVMs are illustrated in figure 6.4 by a single line (labelled “basic lr kernel”) since the profiles are almost identical to each other. Compared to the GMM-LR baseline there is a small increase in accuracy. The change in the average EERs are from 6.12% of the GMM-LR to 5.55%: approximately 10% relative drop in the average EER. However, at low false acceptance rates there is no significant gain compared to the GMM-LR. A much more significant reduction in the error rate is obtained by spherically normalising the kernel. Again the two variations (with and without argument) of the spherically normalised LR kernel are almost identical and are represented by a single line on the DET (labelled “sp norm lr kernel”). Since the dimensionality of the likelihood ratio score-space is six times greater than the likelihood score-space then there is more potential that the Hessian may be ill-conditioned. Spherically normalising the LR kernel has improved upon the accuracy of the underlying GMM-LR baseline at all points on the DET by preconditioning the Hessian. The relative reduction in the average EER obtained by spherically normalising the LR kernel is 34% compared to the GMM-LR baseline is 34% and a reduction of 27% compared to the
6.3. SVM evaluations

Table 6.3: The effect of the likelihood score on score-space classifiers

<table>
<thead>
<tr>
<th>Classifier</th>
<th>% HTER (s.d.)</th>
<th>% EER (s.d.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fisher kernel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>without argument &amp; sph. norm</td>
<td>6.509 (4.194)</td>
<td>6.872 (4.224)</td>
</tr>
<tr>
<td>with argument &amp; sph. norm</td>
<td>6.472 (4.221)</td>
<td>6.868 (4.266)</td>
</tr>
<tr>
<td>LR kernel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>without argument &amp; sph. norm</td>
<td>3.716 (4.406)</td>
<td>4.032 (4.638)</td>
</tr>
<tr>
<td>with argument &amp; sph. norm</td>
<td>3.706 (4.409)</td>
<td>4.029 (4.673)</td>
</tr>
</tbody>
</table>

unnormalised LR kernel.

A similar reduction of 26% in the false rejection rate is observed at low false acceptance rates. In speaker verification systems a low false acceptance rate corresponds to a low probability in misclassifying an impostor. From the DET curves when the probability of misclassifying an impostor is 0.1% the GMM-LR baseline has a probability of 45.5% of rejecting a client but the LR kernel SVM has a lower probability of 35.5%. These probabilities give an indication of how a client might view a system’s usability at a given false rejection rate. The client would observe far fewer rejections on the LR kernel SVM system than the GMM-LR system: roughly speaking approximately one out of every two client test utterances are rejected by the GMM-LR but only one in every three by the LR kernel SVM.

There is a small improvement in the performance (more observable in the HTER than the EER) of the SVM when including the log likelihood score of the GMM as an additional feature in score-space. However, the reduction in the error rate can only be seen in the second or third decimal place (table 6.3). Such a small improvement is statistically insignificant. Therefore, while it does not hurt to include the argument, it is not possible to conclude that the GMM output score encodes information that the derivatives do not.

A major concern for anyone wishing to build a realisation of score-space based SVM systems is the size of the solutions. Table 6.1 reports the best SVM solutions as having nearly a thousand support vectors. The calculations given above indicate that such solutions require over 700 MB of storage space per speaker. Fortunately, the solutions may be reduced to a more reasonable size because a linear SVM is used. The output of a linear SVM is given by equation (4.10),

\[
f(x) = \sum_{s} y_{s}c_{s}(x_{s} \cdot x) + b \quad (6.3)
\]

\[
= \left(\sum_{s} y_{s}c_{s}x_{s}\right) \cdot x + b \quad (6.4)
\]

\[
= w_{0} \cdot x + b \quad (6.5)
\]

where \(s\) enumerates the support vectors. Thus all support vectors may be represented by a single resultant vector \(w_{0}\). The actual number of parameters required per speaker is about four times that of the underlying GMM; the lengths of \(w_{0}\) and the mean and diagonal covariance vectors, for whitening the score-space, plus the number of parameters in the GMM plus one, for the SVM bias.

6.3.3 Overall ranking

The overall ranking of the SVM classifiers on PolyVar can be seen in figure 6.5, which shows the best performing classifiers from the polynomial kernel, Fisher kernel and LR kernel SVMs along side the three
baseline classifiers. The ranking roughly matches that of the YOHO experiments. Starting with the worst classifier: The unnormalised polynomial kernel SVM is the worst performer (not shown in the figure), for which the reasons are now known. The GMM baseline system, which uses only the client model, is next followed closely by the higher order spherically normalised polynomial kernel SVMs (which can be improved by increasing the amount of training data). The log-likelihood score-space (or Fisher) kernel SVM follows, then the GMM-LR baseline system, which was capable of achieving 0% average EER on YOHO. The GMM-LR/SVM system is difficult to place since it is a version of the GMM-LR optimised to work best at a specific false accepts to false rejects ratio. Assuming that the GMM-LR/SVM can be optimised to achieve a better performance than the GMM-LR at the desired false accept to false reject ratio then the GMM-LR/SVM is ranked second. An important result is that at low false rejection probability (low chance of misclassifying an impostor) the spherically normalised polynomial kernel is ranked second. The out-right winner, which was tested on PolyVar and not on YOHO, is the log-likelihood ratio score-space kernel SVM that incorporated spherical normalisation.

The unchanged ranking is an interesting result and shows that moving to a database containing more noisy data does not affect the SVMs much more adversely than the baseline classifiers in terms of performance. In terms of the size of the SVM solutions the number of parameters in the score-space kernel SVMs will always be about four times the number of parameters of the underlying generative model. The effect on the size of the polynomial kernel SVMs is more difficult to quantify since it is clear from table 6.2 that the performance and number of support vectors is linked to the amount of quantisation applied to the training data.

6.4 Conclusion

Polynomial and score-space kernel SVMs were evaluated on the PolyVar database. PolyVar is a much more difficult task than YOHO and has a well defined training and testing protocol. Also the data is noisier and there are about ten times more client tests. The results from this evaluation are therefore more meaningful and more reliable than the results on YOHO. The score-space SVM methods have been proven to perform well on the more difficult PolyVar task. The ranking of the different classifiers remains unchanged from YOHO. This is an important result for the SVM and shows that they are still reliable when used on noisy data.

The results of this chapter validate the conclusions in chapter 5. Spherical normalisation has proven to be a valuable method for improving the performance of SVMs. Just as on YOHO, the new normalisation method enables polynomial SVMs to achieve significantly better results than would otherwise be obtained. It has also been verified that the likelihood score-space (Fisher) kernel SVM yields a significant improvement over the GMM baseline upon which the kernel is derived. The evaluation of the likelihood ratio score-space (LR) kernel was performed successfully. Most significantly, the major improvements in the LR kernel was achieved with the use of spherical normalisation, reinforcing the effectiveness of the new normalisation. Spherical normalisation does not add any extra information but merely enabled the optimiser to find a better solution. It is worth mentioning that it is not always easy to set the parameter in spherical normalisation: in some preliminary experiments on pair HMMs (section 7.3) the parameter was set by searching the parameter space so that the SVM objective function was minimised.

The significant improvement in the accuracy achievable with a generative model by applying the score-space kernel SVM strategy is an illustration of the effectiveness of combining generative models with discriminative classifiers. The generative model was exploited to create a fixed-size representation of a
variable length sequence so that a discriminative classifier (the SVM) could discriminate between complete sequences. One motivation for using a generative model is that it is a full probability model of the underlying processes that should retain as much information as possible in the mapping to the score-space. This enabled discrimination to be performed at the utterance-level. The alternative, which is frame-level discrimination, is less effective since the discriminative classifier may discard information that is deemed insignificant to the classification of the frame but is actually still significant to the classification of the sequence. This point is proved by the fact that frame-level discriminative classifiers are out-performed by generative models on speaker verification (see section 3.5.1) but an utterance-level discriminative classifier is better still.

By building a generative model into the discriminative SVM framework and with the necessary normalisations applied, we have reduced the error rate on the PolyVar speaker verification task significantly.
Chapter 7

Conclusion

7.1 Summary

In this thesis we studied the use of support vector machines (SVMs) for text independent speaker verification. Two main approaches were considered. The first is a frame based approach using polynomial kernels to score the frames of an utterance separately. The second approach is an utterance based approach using score-space kernels to score complete utterances directly without scoring individual frames.

State-of-the-art speaker verification systems rely on generative models to recognise speakers. Discriminative classifiers such as radial basis function (RBF) networks and multilayer perceptrons (MLPs) have been found to perform less well than generative models such as Gaussian mixture models (GMMs). The approach using polynomial kernel SVMs, which is similar to the RBF and MLP approaches, also proved to be less successful than well trained GMMs. It is a curious result since discriminative approaches for classification should in theory be better than generative ones since the former are optimised to minimise the classification error rate explicitly whereas the latter are not. Generative models that have been made more discriminative by a simple ratio method have been used successfully.

GMMs are usually the workhorses in speech systems. Discrimination in GMMs may be achieved through the use of mutual information (or frame discrimination) objective functions to adjust the model parameters discriminatively. In hidden Markov models (HMMs) discrimination may be achieved by using MLP to estimate state probabilities in the place of GMMs. Either way discrimination occurs at the frame level. Current discriminative approaches achieve sequence classification by scoring individual frames of data discriminatively but do not consider the sequence as a whole. In speaker verification the final score assigned to an utterance is much more important than the individual frame scores and yet discriminative classifiers are optimised to minimise the number of errors on the frame-level. This is not optimal. Discriminative classifiers discard information that is not seen as relevant to the classification. When the discrimination is at the level of individual frames, the focus is on minimising the errors when scoring one frame at a time: information useful for determining the classification of the sequence could be discarded inadvertently.

We showed that the current performance of speaker verification systems is improved significantly by incorporating generative models into a discriminative framework of an SVM so that an entire utterance is classified discriminatively instead of the constituent frames. Specifically, score-space kernel [SGN01] (which includes the Fisher kernel [JH99]) SVMs were used and significant reductions in the error rates were observed on the PolyVar database. The adopted strategy differs from many of the existing methods. such
as the use of the mutual information (or frame discrimination) objective function to adjust the parameters of a generative model discriminatively or the use of MLPs to discriminatively estimate the probabilities of an HMM. In both these cases the discrimination occurs at the frame-level. Instead, the SVM score-space kernel method uses a GMM to create a representation of an utterance in a much higher dimensional space (called the score-space) of some fixed dimensionality. Each processed sequence is treated as a single point in that space for further processing by the SVM. This makes the discrimination between whole sequences more explicit, simultaneously allowing the underlying generative model to estimate the true likelihood distribution: a property that may be desirable in speaker verification. With this SVM approach, exploiting a good generative model should retain as much information as possible until the utterance-level decision is made.

In this thesis we developed the techniques to overcome the difficulties arising from an application of SVMs to speaker verification. We demonstrated the effectiveness of SVMs for use in speaker verification when an appropriate of kernel is chosen and correct normalisation is used. The techniques were developed on the YOHO database and then evaluated on the more difficult PolyVar database. This separation of the development from the evaluation is important to ensure that the methods are general and that the classifiers have not been tuned to one particular database. The evaluation in chapter 6 of the likelihood ratio score-space kernel SVM reduced error rates on PolyVar by about 34% relative from the GMM-LR baseline and by 32% from the current state-of-the-art (GMM-LR/SVM) classifier, while using a reasonable number of parameters. To achieve this result two different normalisation methods had to be applied. The first is whitening which involves normalising the vectors in the score-space so that they are distributed with zero mean and unit diagonal covariance. This is an important step: without it the SVM becomes biased towards the components of the score-space with the highest variances. The second normalisation is a new technique called spherical normalisation that preconditions the SVM’s Hessian enabling the optimiser to converge to a better solution.

Spherical normalisation (chapter 5) is the major novel contribution of this thesis. The method involves a simple mapping from a hyperplane to a unit hypersphere in a higher dimensional space. It was initially developed as a fix to prevent the polynomial kernel from generating large values in the Hessian by limiting the range of values that the elements of the matrix may take and making it diagonally dominant. This was done by exploiting the fact that dot products between unit vectors are well scaled. It was realised that this mapping imparts upon the polynomial kernel properties similar to radial basis functions. More significantly it has potential to be applied to any existing kernel function. This includes kernels for which the explicit form of induced non-linear transformation is unknown. Applying spherical normalisation together with the polynomial kernel can impart RBF properties upon any kernel function. This ability makes spherical normalisation a very powerful tool for researchers trying to develop new kernels.

The effectiveness of spherical normalisation is demonstrated by experimental results. Section 7.3 outlines some preliminary work on a different type of sequence kernel called the pair HMM kernel [Wat99]. In a speaker verification task this kernel almost always yields a Hessian that is so badly scaled that it is virtually unusable without spherical normalisation. Applying the normalisation enabled SVMs to be trained much more easily. The results obtained on YOHO and PolyVar are much better because of spherical normalisation. In the polynomial kernel experiments spherical normalisation reduced the error rates and enabled higher order polynomials to be used. Without spherical normalisation the 34% reduction in the PolyVar equal error rates by the likelihood ratio score-space kernel would not have been possible.
7.2 Future work: spherical normalisation

There are many avenues for further research on spherical normalisation. For example, the most obvious one is its application to other kernels such as the pair HMM (see section 7.3). Also, projections for mapping from plane to sphere other than the ones listed in figure 5.3 exist that may be investigated. One need only look at some of the projections used by cartographers: projections used to make maps of the Earth, of which there are thousands. In our case we are interested in the reverse of these projections extended to arbitrary dimension.

Conformality, equidistance and equal-area are three additional properties that may be enforced on spherical projections, which may also lead to other interesting properties. Take the equidistant projection for example. It is one which has the property that the arc length between one fixed point and any other point on the sphere is the same as the corresponding Euclidean separation in the plane. Since the angle between two vectors on a sphere is proportional to the length of arc between, then the RBF interpretation of spherically normalised kernels is the cosine of the Euclidean separation in the plane (but only if one of the vectors corresponds to the aforementioned fixed point).

Spherical normalisation may be interpreted as a new kernel in itself. Figure 5.4 is essentially an illustration of the different ways that the new kernel can score pairs of vectors. Further investigation into the effects of assigning different values to the parameters of this kernel (the spherically normalised general polynomial kernel, equation 5.14) is required. In section 5.3.2 the problem of setting these parameters to guarantee a valid kernel function was left unsolved.

Moving away from SVMs and kernels, further comparisons with the RBF yield more uses for spherical normalisation. The mathematical form of the RBF kernel is none other than the Mahalanobis distance between two points in space — a commonly used statistic. As a consequence, the mathematical formulation of a spherically normalised polynomial kernel with appropriate values for its parameters may be viewed as a distance measure that is based upon the angles between vectors. A possible application for this measure may be found in signal processing. The cross-correlation between two signals gives an indication of the similarity between them. If the cross-correlation is large then it is likely that the two signals are similar. The cross-correlation is effectively a dot product between two signals. Using the spherical normalisation equations enables the conversion of the cross-correlation score to a similarity measure analogous to the Mahalanobis distance. The analogies may even be extended to spherical Gaussians!!

7.3 Future work: Pair HMMs

Score-space kernels are not the only way to exploit generative models to classify sequences. In this section we review the pair HMM model, describe how it can be used to estimate dot products and present some preliminary results on speaker verification.

7.3.1 Introduction

The pair HMM is a hidden Markov model that has been adapted to model pairs of sequences. It was developed by Durbin et. al. [DEKM98] as a model of DNA alignment where the states occur in pairs. It takes into consideration the variability in the length of the sequences and is capable of handling missing data where sections of sequences may be unknown.

The pair HMM can be interpreted as a form of statistical dynamic time warping (DTW). DTW has been used successfully in speech recognition, in particular the earlier speech systems before HMMs. DTW still
has its benefits over HMMs. As a simple example, in speaker verification systems where only one phrase is available as training data a DTW system may be applied directly while the parameters of an HMM cannot be learned reliably from so little data easily (we do not mean that DTW outperforms HMMs in such situations but rather that DTW has its benefits).

DTW works by aligning a test sequence with a sample sequence and assigning a score or distance measure to the alignment. In principle, if the alignment of some sequence A against another sequence B is guaranteed to give the same result as aligning B against A then we may begin to examine the alignment score as a possible candidate for an inner product estimate. A kernel function derived from a DTW type algorithm will allow SVMs to classify variable length sequences.

The pair HMM works by aligning two sequences on each other while imposing a conditional symmetry on the process. The symmetry allows the likelihood score of the pair HMM to be interpreted as a dot product in some space. We shall begin by describing the pair HMM topology and the training and testing algorithms, then we shall show how they may be combined with an SVM.

### 7.3.2 The pair-wise alignment model

A pair HMM is a hidden Markov model that generates two sequences of symbols simultaneously. Let the two sequences, which need not be of the same length, be labelled A and B. An illustration of the pair HMM model is shown in figure 7.1a. It consists of a match state, \( q_{AB} \), and two insert states, \( q_A \) and \( q_B \). The match state emits a symbol into both sequences A and B simultaneously (they need not be the same symbol). The insert state \( q_A \) emits only into sequence A. It is a state that generates insertions in sequence A leaving gaps in sequence B. State \( q_B \) is defined similarly.

Making transitions between the doubly emitting match state, and the singly emitting insert states allows gaps to be inserted into either sequence at any time. This is where the analogy with DTW arises. Figure 7.1b illustrates this with a simple example showing the state occupation sequence and the corresponding alignment.

By constraining the transition probabilities in the pair HMM so that the exit probabilities from each state are normalised it is possible to parameterise them all using a few variables. Let the probability of making a transition from the match state to an insert state be denoted by \( \delta \), the probability of staying in the insert state be \( \varepsilon \) and the probability of making a transition to the end state, \( q_F \), be denoted \( \tau \). Typically the value of \( \delta \) is small in order to penalise insertions. The transition probabilities can be summarised as follows,

\[
P(q_A|q_A) = P(q_B|q_B) = P(q_B|q_{AB}) = P(q_A|q_{AB}) = \delta \quad (7.1)
\]

\[
P(q_A|q_A) = P(q_B|q_B) = \varepsilon \quad (7.2)
\]

\[
P(q_F|q_A) = P(q_F|q_B) = P(q_F|q_{AB}) = P(q_F|q_I) = \tau \quad (7.3)
\]

\[
P(q_{AB}|q_A) = P(q_{AB}|q_B) = 1 - \varepsilon - \tau \quad (7.4)
\]

\[
P(q_{AB}|q_{AB}) = P(q_{AB}|q_I) = 1 - 2\delta - \tau \quad (7.5)
\]

Adjusting these parameters enables the model to vary the penalties it applies to the differences in the lengths of the sequences and the number of insertions that are made.

A set of matrices (or a density model in the case of continuous data) provide emission and transition probability estimates. The emission probability for state \( q_{AB} \) is the joint probability of emitting two symbols simultaneously. This probability may not be straight forward to estimate without some approximations (see section 7.3.4).
7.3. Future work: Pair HMMs

a: The pair hidden Markov model for pairwise alignment of two sequences.

b: One possible alignment of pair HMM given a template sequence and a test sequence. The corresponding state occupation sequence is shown for this particular alignment.

Figure 7.1: The pair hidden Markov model
7.3.3 The forward and backward recursion

The forward, $\alpha$, and backward, $\beta$, recursions can be derived using the same method described in section 3.2.3. The main difference is that the recursion must be a function of the time index in two sequences instead of just one. Let $\alpha_{t_A,t_B}(k)$ denote the value of the forward recursion for state $q_k$, where $k \equiv \{AB, AB, B\}$, at time $t_A$ in sequence $A$ and $t_B$ in sequence $B$. In the case of the doubly emitting state the recursion at time $t_A$ and $t_B$ are dependent upon the states occupied at the previous time $t_A - 1$ and $t_B - 1$. The forward recursion for this state is,

$$\alpha_{t_A,t_B}(AB) = \sum_{x_{t_A-1:t_B-1}} \alpha_{t_A-1:t_B-1}(k) P(q_{AB} | q_{t_A:t_B}) P(x_{t_A}, x_{t_B} | q_{AB})$$

(7.6)

The singly emitting state $q_A$ depends only on the states occupied at time $(t_A - 1)$ and time $t_B$ since it emits only into sequence $A$ and not into $B$. Similar dependencies apply to $q_B$. The forward recursions for these states are,

$$\alpha_{t_A,t_B}(A) = \sum_{x_{t_A-1:t_B-1}} \alpha_{t_A-1:t_B-1}(k) P(q_A | q_{t_A}, x_{t_B}) P(x_{t_A} | q_A)$$

(7.7)

$$\alpha_{t_A,t_B}(B) = \sum_{x_{t_A-1:t_B-1}} \alpha_{t_A-1:t_B-1}(k) P(q_B | q_{t_A}, x_{t_B}) P(x_{t_B} | q_B)$$

(7.8)

initialised by,

$$\alpha_{0,t_0}(AB) = 1$$

(7.9)

and all other $\alpha_{t_i,0}$ and $\alpha_{0,t_j} (i \neq 0)$ equal to zero. Lastly, the termination is,

$$\alpha = \sum_{k} P(q_f | q_k) \alpha_{T_A,t_B}(k)$$

(7.10)

Similarly the backward recursions are: for the doubly emitting state,

$$\beta_{t_A,t_B}(AB) = \beta_{t_A+1,t_B+1}(AB) P(q_{AB} | q_{t_A:t_B}) P(x_{t_A+1:t_B+1} | q_{AB})$$

(7.11)

$$+ \beta_{t_A+1,t_B}(A) P(q_A | q_{t_A}, x_{t_B}) P(x_{t_A+1} | q_A)$$

$$+ \beta_{t_A+1,t_B+1}(B) P(q_B | q_{t_A}, x_{t_B}) P(x_{t_B+1} | q_B)$$

the singly emitting states,

$$\beta_{t_A,t_B}(A) = \beta_{t_A+1,t_B+1}(AB) P(q_{AB} | q_A) P(x_{t_A+1:t_B+1} | q_{AB})$$

(7.12)

$$+ \beta_{t_A+1,t_B}(A) P(q_A | q_{t_A}, x_{t_B}) P(x_{t_A+1} | q_A)$$

$$\beta_{t_A,t_B}(B) = \beta_{t_A+1,t_B+1}(AB) P(q_{AB} | q_B) P(x_{t_A+1:t_B+1} | q_{AB})$$

(7.13)

$$+ \beta_{t_A+1,t_B+1}(B) P(q_B | q_{t_A}, x_{t_B}) P(x_{t_B+1} | q_B)$$

which are initialised by

$$\beta_{t_A,t_B}(k) = p(q_f | q_k)$$

(7.14)

and

$$\beta_{0,t_B+1}(k) = \beta_{t_A+1,t_B+1}(k) = 0$$

(7.15)

for $k \equiv AB, AB, B$, $i = 1 \cdots T_A - 1$ and $j = 1 \cdots T_B - 1$. No specific termination is required since $t_A + t_B \geq 1$ in the backward recursion.
A pair HMM can be trained by using the maximum likelihood criterion and the EM algorithm using
the forward and backward recursions in a similar fashion to the standard HMM training approach outlined
in 3.2.3.

The likelihood of an alignment can be computed using the forward recursion by evaluating
\[
P(X^A, X^B | M) = \alpha_{r_n \tau_n}(q_f)
\]  
(7.16)
for the end state. The Viterbi alignment algorithm for pair HMMs can be obtained by replacing the
summations in the forward and backward recursions with a max function. This approximation allows the pair
HMM recursions to be computed more quickly.

### 7.3.4 The pair HMM as a kernel function

The ability of the pair HMM to align two variable length sequences and to assign them a single match score
is the first step in obtaining a kernel function. The match score must still be proven to estimate dot products
in some space. This was done in two steps by Watkins [Wat99]. It is first shown that if a joint probability
density is conditionally symmetrically independent then the joint probability score is a dot product in some
space. Then it is shown that the joint probability estimated by the pair HMM is conditionally symmetrically
independent.

**Conditional symmetric independence**

A joint probability distribution is said to be conditionally symmetrically independent if it is a mixture of
a finite number of symmetric independent distributions. This means that, when the joint probability is
conditioned upon one of the mixtures then it will have a symmetric independent distribution (that of the
conditional mixture).

**Conditional symmetric independence kernels**

It can be shown that a joint probability distribution that is conditionally symmetrically independent is a dot
product in some space. Let \( X \) and \( Z \) be two random variables. The joint probability of \( x \) and \( z \) drawn from
the random variables \( X \) and \( Z \) respectively is \( P(x, z) \). By imposing symmetry,
\[
P(x, z) = P(z, x)
\]  
(7.17)
for all \( x, z \). Introduce a third discrete random variable \( C \) such that
\[
P(X, Z | C) = P(X | C)P(Z | C)
\]  
(7.18)
so that
\[
P(x, z | c) = P(x | c)P(z | c)
\]  
(7.19)
By constraining the distributions such that given \( c \) was drawn from random variable \( C \) the distributions of
\( X \) and \( Z \) are identical then,
\[
p(x, z) = \sum_c p(c | c)p(c | e)p(c)
\]  
(7.20)
\[
= \sum_c \left( p(x | c) \sqrt{p(c)} \right) \left( p(z | c) \sqrt{p(c)} \right)
\]  
(7.21)
By defining the feature space mapping as,

\[ \Phi(x) = \langle p(x|c) \sqrt{p(c)} : c \in C \rangle \] (7.22)

the joint probability can be expressed as the dot product

\[ p(x,z) = \Phi(x) \cdot \Phi(z) \] (7.23)

**Conditionally symmetric independent pair HMMs**

There are three conditions that must be satisfied in order for a pair HMM to estimate conditionally symmetrically independent joint probabilities over a pair of sequence of symbols [Wat99].

1. The joint probability distribution is unchanged when the state labelled \( q_k \) is relabelled \( q_l \) and vice versa. This is a requirement of symmetry so that interchanging the two sequences aligned on a pair HMM does not change the likelihood of the model. Without such a constraint, different insertion probabilities resulting from the two insertion states will destroy any symmetry.

2. The pair HMM has independent insertion property. That is, any sequence of symbols inserted into sequence \( A \) by state \( q_k \) is independent of the symbols inserted into sequence \( B \) by \( q_l \).

3. For all doubly emitting (match) states the pair-wise-symbol-emission joint probability distributions must be conditionally symmetrically independent.

The proof by Watkins [Wat99] is as follows. Consider the substitution illustrated in figure 7.2a. Here the match state is replaced by a network of states: a non-emitting start state, \( s_l \), a set of emitting atomic states, \( s_0, s_1, \cdots \) and a non-emitting end state, \( s_F \). Each of the atomic states have symmetric independent probability distributions. The distribution of symbols emitted by the network are identical to those emitted by the substituted match state. From this decomposition it follows that the match state, consisting of a mixture of symmetric independent joint probability distributions, must have a conditionally symmetrically independent joint probability distribution.

Consider one instance of a pair HMM alignment. Let \( m = \{ m_0, \cdots, m_n \} \) denote the sequence of states occupied in \( q_{AB} \) in this instance. In going from \( m_{i-1} \) to \( m_i \) the process may pass through a series of states in either \( q_A \) or \( q_B \). Let these sequences of insertion states be denoted by \( a_i \) and \( b_i \) respectively. Let \( a \) denote the complete set of insertions into sequence \( A \), \( a = \{ a_0, \cdots, a_n \} \) and let \( b \) be defined similarly. Finally, let \( A(m_i) \) and \( B(m_i) \) denote the symbols emitted into sequences \( A \) and \( B \) by state \( m_i \). The notation is illustrated in figure 7.2b.

From the independent insertion property,

\[ P(a_i, b_i | m_i, m_{i+1}) = P(a_i | m_i, m_{i+1})P(b_i | m_i, m_{i+1}) \] (7.24)

for \( 0 \leq i \leq n \), so that

\[ P(a, b | m) = \prod_{i=1}^{n} P(a_i, b_i | m_i, m_{i+1}) \] (7.25)

\[ = P(a | m)P(b | m) \] (7.26)

Since \( m_i \) is an atomic state with an independent emission distribution,

\[ P(A(m_i), B(M_i) | m_i) = P(A(m_i) | m_i)P(B(M_i) | m_i) \] (7.27)
7.3. Future work: Pair HMMs

a: Rewriting the doubly emitting match state as a set of atomic states.

b: An instance of a pair HMM.

Figure 7.2: The pair HMM state substitution and notation.
for $0 < i \leq n$. Since state $q_i$ does not affect symbols in sequence $B$ and vice versa,

$$P(A, B|a, b, m) = P(A|a, m)P(B|a, m).$$  \tag{7.28}

Summing over $a$ and $b$ gives

$$P(A, B|m) = \sum_{a,b} P(A, B|a, b, m)P(a, b|m).$$  \tag{7.29}

Substituting equations (7.26) and (7.28) into (7.29) gives

$$P(A, B|m) = \sum_{a,b} (P(A|a, m)P(a|m))(P(B|b, m)P(b|m))$$

$$= P(A|m)P(B|m).$$  \tag{7.30}

Hence the pair HMM joint probability distribution is conditionally symmetrically independent.

### 7.3.5 Implementation

In our implementation of the pair HMM for text independent speaker verification the emission probabilities of the states were estimated using the Gaussian mixture models that were created for each speaker in the baseline GMM experiments.

**Emission probabilities**

Given the GMM of a particular speaker, the corresponding pair HMM emission probabilities for the insertion states are the likelihood estimates of the GMM generating the feature vector. The match state emission probability is a joint probability that two symbols are emitted. In general the full joint probability is difficult to estimate. However, in order that the pair HMM is a valid kernel it must estimate conditionally symmetrically independent joint probabilities. This simplifies the problem. The match state of a pair HMM used for text independent speaker verification has just one atomic state. This atomic state emits the feature vectors corresponding to the modelled speaker independently of the transcription. Therefore, the match state emission probability must be the product of the likelihood estimates of the aforementioned GMM generating each of the two observation vectors that are emitted by this state.

**Viterbi approximation**

Using the pair HMM as a kernel function requires approximately $\frac{1}{2}N_{\text{training}}^2$ pair-wise alignments to be computed in order just to calculate the Hessian for training an SVM. To test each SVM, an additional $(N_{\text{support vectors}} \times N_{\text{test utterances}})$ alignments must be computed.

When computing the product over a large number of likelihoods it is easy to generate a numerical overflow were the values can no longer be stored in a double precision floating point variable. To overcome this problem all computation is done in the log-domain. However, performing summations in the log-domain is computationally expensive. Coupling this with the large number of alignments that must be computed means that the full alignment is currently too computationally expensive. The Viterbi approximation is used to decrease the amount of computation required by the implementation. It replaces the sum in the forward and backward recursions with a max function.
7.3. Future work: Pair HMMs

Utterance length normalisation

For a text independent pair HMM the dependence upon the lengths of utterances with different transcriptions must be reduced as much as a possible. The most significant contribution to the pair HMM output probability comes from the emission probabilities. The emission probabilities can be normalised by dividing the log-likelihood of the emission probabilities by the length of the sequence into which the symbol is emitted. This effectively converts the total sum over the log emission probabilities to a mean over the log emission probabilities in a similar fashion to the baseline GMM described earlier.

Transition probabilities

The transition probabilities of the pair HMM are defined by three parameters, \( \delta, \varepsilon \) and \( \tau \). It is vital that these parameters be set properly depending on the lengths of the utterances, otherwise alignments consisting of two long utterances will be penalised by the transition probabilities significantly more than alignments of two short utterances.

The product over the emission probabilities is independent of the number of transitions made between match and insert states since all of the states are modelled by a single GMM. As a result, only one transition needs to be made to the insertion state to compensate for the difference in length of the two utterances. For two utterances of length \( n_A \) and \( n_B \) with \( n_B > n_A \) the simplest state sequence given this particular implementation of the is \( q_I \) followed by \( n_A \) states of \( q_{AB} \), then \( (n_B - n_A) \) states of either \( q_A \) or \( q_B \) and finally \( q_F \).

Given that the Viterbi approximation is used, the product over the transition probabilities of the proposed state sequence is, \( \prod (1 - 2\delta - \delta \varepsilon - \tau) \). We shall maximise the probability of this path. Differentiation with respect to \( \delta \) and equating the result to zero gives,

\[
\delta = \frac{1 - \tau}{2(n_A + 2)} \tag{7.32}
\]

Likewise for \( \tau \),

\[
\tau = \frac{1 - 2\delta}{n_A + 2} \tag{7.33}
\]

Substituting (7.32) into (7.33) allows both \( \delta \) and \( \tau \) to be calculated. The probability of reaching the final state, \( P(q_F|q_{a}) = \tau \), could be intuitively set to the reciprocal of the length of one of the utterances. Eliminating \( \delta \) from (7.33) gives the result, \( \tau \approx 1/n_{A} \).

The remaining parameter, \( \varepsilon \), applies a penalty to utterances of different lengths. The greater the difference in length the smaller the probability of a match. Since the transition probabilities are constrained,

\[
0 < P(q_{AB}|q_{a}) < 1, \\
0 < 1 - \varepsilon - \tau < 1, \\
0 < \varepsilon < 1 - \tau. \tag{7.34}
\]

In this implementation for text independent speaker verification the penalty applied by differences in length is minimised by setting \( \varepsilon \) to its upper limit.

Spherical normalisation

From an examination of the numerical values generated by the pair HMM it can be seen that the Hessian will be ill-conditioned. Recall the polynomial kernel experiments. Initially results were poor but the
7.3. Future work: Pair HMMs

problem was tracked down to an ill-conditioned Hessian. Spherical normalisation, equation (5.16), was
developed to fix the problem. The same formula can be applied here — the only difficulty lies in the
selection of an appropriate normalising constant, $d$.

The normalising constant for the polynomial kernel could be determined by analysing the geometry the
spherical normalisation mapping. The input data existed in a fixed 24 (for the YOHO database) dimensional
space and that needed to be mapped onto the surface of a 25 dimensional unit sphere. It was important that
the data was mapped onto the sphere without it being clumped up near the pole (when $d$ is too large) or
spread in a thin line around the equator (when $d$ is too small). Each component of the input data had been
normalised to zero mean and unit variance therefore a normalising constant of one was deemed adequate.

In the case of the pair HMM kernel, the input data to the kernel are variable length sequences that
cannot be analysed in the same way. To estimate the normalisation constant we perform a numerical
search. Taking a small random sample of one hundred utterances equally divided between the user and
the impostors train two SVMs using the normalised pair HMM kernel with two extreme values of $d$ and
compute the objective function at the solution. Since there are only one hundred samples training takes just
a few seconds. By training another SVM with a normalisation constant midway between the two extremes
we can home in on a value that will minimise the objective function given this particular subset. Choosing
a different subset of data affects the best value of $d$ only slightly. Indeed we observed that the estimated
normalisation factor does not vary significantly between different user/impostor sets.

7.3.6 Preliminary result and discussion

The equal error rate obtained by this evaluation was 1.05%. The corresponding result obtained on the
GMM baseline was 1.08%. The pair HMMs have, on average, 49 support vectors in their solutions. The
combination of the discriminative SVM with the generative pair HMM model has yielded a small improve-
ment in the equal error rate but not enough to rival any of the other techniques described in this thesis. The
reason is quite simple.

Examine equations (7.20) to (7.22) and (7.31). The pair HMM is a valid kernel by having a symmetri-
cally independent joint probability distribution when conditioned on the sequence of atomic states (7.31).
From equation (7.20), the joint probability must be marginalised over the possible sequence of atomic
states. Thus each dimension of the feature space, equation (7.22), corresponds to one possible sequence of
the atomic match states.

The implementation of the pair HMM used in this evaluation was for text independent speaker verifica-
tion. Therefore, there can be only one atomic state corresponding to a match between the two emitted
symbols. This means that the pair HMM is mapping to a feature space of just one dimension. The implica-
tion is that any benefit of a discriminative classifier is immediately made redundant since a one dimensional
feature space problem can be solved by applying a threshold or hypothesis testing. An SVM is capable
of mapping the one dimensional feature space of the pair HMM to higher dimensions by further applying
another kernel. The implementation used spherical normalisation that mapped to a two dimensional space.
Thus the final result is not one that can be obtained by applying a threshold. This may explain the slight
improvement in performance.

It is suspected that without the SVM’s ability to map to higher dimensional spaces and to combine the
result of multiple alignments on the support utterances there would be no additional boost in performance
over the standard HMM. It is therefore clear that the pair HMM is not a good model for use with text
independent speaker recognition tasks. It may be more suited to text dependent tasks where there is more
than one atomic state. Each atomic state representing a unit of speech, for example. The extra fixes
used in the implementation, such as utterance length normalisation, will not be required and a proper duration model can be incorporated. Furthermore, the Viterbi alignment was used to reduce computation requirements. In doing so the pair HMM is again restricted to just one sequence of atomic states, reducing the feature space dimensionality to one. In a pair HMM implementation that has multiple atomic states it may be important to take this into consideration. A full alignment may be an absolute requirement.

We believe that the pair HMM has potential as an SVM kernel. Further research should be done on a more appropriate task such as text dependent speaker verification.
Bibliography


[Soo90] F. Soong. A vector quantisation approach to speaker recognition. AT


Index

backward recursion
  HMM, 23
  pair HMM, 92

chunking, 49

data space, 41
discriminative model, 14

EER, 10
entropy, 30
equal error rate, see EER

feature space, 41
forward recursion
  HMM, 21
  pair HMM, 92

generative model, 14
GMM, 17
GMM-LR, 18, 28
GMM/SVM, 29

half total error rate, see HTER
HMM, 19
HTER, 10
hypothesis testing, 9

input space, 41
input vector, 34
kernels, 43–44
  Fisher, 46, 69, 82
  likelihood, 46, 69, 82
  likelihood ratio, 46, 69, 82
  normalised polynomial, 64, 74
  pair HMM, 93–96
  polynomial, 45, 58, 74
  RBF, 45, 50

score space, 45, 82
  Vovk’s polynomial, 45

Lagrange multiplier, 36
LPC, 6

margin
  maximising, 36
  soft, 38
Mercer’s condition, 44
MLP, see neural network
mutual information, 30

neural network, 26–27
  MLP, 26, 29
  recurrent, 27
  time delay, 27
Perceptual linear prediction, 7
PLP, 7
polynomial
  classifier, 24–25, 56–60, 67
  expansion, 58
  general kernel, 45
  kernel, 57–60
  normalised kernel, 64, 60–68
  Vovk’s kernel, 45
PolyVar, 12

radial basis function, see RBF
RBF
  kernel, 45, 50–51, 64
  network, 27–28, 50–51
risk, 34–35, 50
  empirical, 35
  regularised, 35, 50
score space, 45
  likelihood, 46
likelihood ratio, 46
slack variable, 38
spherical normalisation, 60–64
    kernel, 67
support vector, 38
Viterbi, 23, 93, 97
YOHO, 12