A Comparison of Gaussian Mixture Variants with Application to Automatic Phoneme Recognition

Rinus Brand

Thesis presented in partial fulfilment of the requirements for the degree
Master of Science in Electronic Engineering
at the University of Stellenbosch

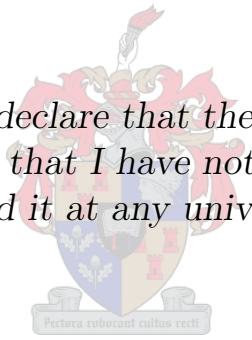
Supervisor: Prof J.A. du Preez

December 2007

Pectora roborant cultus recti

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.



SIGNATURE DATE

Abstract

The diagonal covariance Gaussian Probability Density Function (PDF) has been a very popular choice as the base PDF for Automatic Speech Recognition (ASR) systems. The only choices thus far have been between the spherical, diagonal and full covariance Gaussian PDFs. These classic methods have been used for some time, but no single document could be found that contains a comparative study on these methods in the use of Pattern Recognition (PR).

There also is a gap between the complexity and speed of the diagonal and full covariance Gaussian implementations. The performance differences in accuracy, speed and size between these two methods differ drastically. There is a need to find one or more models that cover this area between these two classic methods.

The objectives of this thesis are to evaluate three new PDF types that fit into the area between the diagonal and full covariance Gaussian implementations to broaden the choices for ASR, to document a comparative study on the three classic methods and the newly implemented methods (from previous work) and to construct a test system to evaluate these methods on phoneme recognition.

The three classic density functions are examined and issues regarding the theory, implementation and usefulness of each are discussed. A visual example of each is given to show the impact of assumptions made by each (if any).

The three newly implemented PDFs are the Sparse-, Probabilistic Principal Component Analysis- (PPCA) and Factor Analysis (FA) covariance Gaussian PDFs. The theory, implementation and practical usefulness are shown and discussed. Again visual examples are provided to show the difference in modelling methodologies.

The construction of a test system using two speech corpora is shown and includes issues involving signal processing, PR and evaluation of the results. The NTIMIT and AST speech corpora were used in initialisation and training the test system. The usage of the system to evaluate the PDFs discussed in this work is explained.

The testing results of the three new methods confirmed that they indeed fill the gap between the diagonal and full covariance Gaussians. In our tests the newly implemented methods produced a relative improvement in error rate over a similar implemented diagonal covariance Gaussian of 0.3–4%, but took 35–78% longer to evaluate. When compared relative to the full covariance Gaussian the error rates were 18–22% worse, but the evaluation times were 61–70% faster. When all the methods were scaled to approximately the same accuracy, all the above methods were 29–143% slower than the diagonal covariance Gaussian (excluding

the spherical covariance method).

Opsomming

Die diagonale kovariansie Gaussiese Waarskynlikheid-Digtheid-Funksie (WDF) is 'n baie populêre keuse as basis vir outomatiese spraak-herkenning sisteme. Tot dusver was die enigste keuses gewees tussen die sferiese-, diagonale- en vol-kovariansie Gaussiese WDFs. Alhoewel hierdie klassieke metodes al vir 'n geruime tyd in gebruik is, kon daar geen dokument gevind word wat hierdie metodes teenoor mekaar opweeg vir die gebruik in patroon herkenning nie.

Daar bestaan ook 'n gaping in terme van kompleksiteit en spoed tussen die diagonale en vol kovariansie modelle. Die verskil in akkuraatheid, spoed en model-grootte tussen hierdie twee metodes is relatief groot. Daar bestaan 'n noodsaaklikheid vir een of meer modelle wat die spasie tussen hierdie twee klassieke metodes kan vul.

Die hoofdoele van hierdie tesis is die evaluasie van drie nuwe WDF tipes wat die area tussen die diagonaal en vol kovariansie Gaussiese implementasies vul om sodoende die keuses van WDF vir outomatiese spraakherkenning groter te maak, om 'n vergelyking te tref tussen al die nuut geïmplementeerde (vanaf vorige werk) en klassieke metodes en dit te dokumenteer, en om 'n toetsstelsel te implementeer wat hierdie metodes op foneemherkenning evalueer.

Die drie klassieke digtheidfunksies word elk ondersoek in terme van hul teorie, implementasie en bruikbaarheid. 'n Visuele voorbeeld van elke metode word voorsien om die impak van die aannames wat deur elk gemaak word (indien enige) voor te stel.

Die drie nuwe voorgestelde WDFs is die yl kovariansie, die waarskynlikheids gebaseerde hoof komponent analise kovariansie en faktor analise kovariansie Gaussiese WDFs. Die teorie, implementasie en praktiese bruikbaarheid van hierdie metodes word gewys en bespreek. Weereens word visuele voorbeelde gebruik om die verskille in die modellering metodieke uit te wys.

Die opstel van 'n toetsstelsel wat twee spraak-databasise gebruik, word geïllustreer, insluitende aspekte rakende seinprosessering, patroonherkenning en evaluasie van die resultate. Die NTIMIT en AST spraak-databasisse was gebruik vir inisialisering en afrigting van hierdie toetsstelsel. Die gebruik van hierdie stelsel om die verskeie WDFs te evalueer word verduidelik.

Toetsing van die drie nuwe metodes benadruk die feit dat hulle inderdaad die gaping tussen die diagonale en vol kovariansie metodes vul. In die verskeie toetse wat uitgevoer was, het die nuut geïmplementeerde WDFs 'n relatiewe verbetering op die fout tempo van 'n soortgelyke diagonale kovariansie Gaussiese WDF getoon van omtrent 0.3–4%. Dit het egter 35–78% langer geneem om te evalueer. Wanneer ons die metodes vergelyk teenoor

die vol kovariansie Gaussiese WDF, kry ons 'n relatiewe verswakking in die fout tempo van 18–22%. Evaluasie was 61–70% vinniger. Met al die metodes geskaleer tot min of meer dieselfde akkuraatheid, was al die bogenoemde metodes 29–143% stadiger as die diagonale kovariansie Gaussiesse WDF (uitsluitend die sferiese-kovariansie Gaussiesse WDF).

Acknowledgements

I would like to thank the following people. Without them, this work would certainly not have been possible:

- Prof. J. A. du Preez, for his advice, insights and ideas, for being generally interested in my work and having the ability to always keep me motivated,
- my parents, who not only supported me financially, but also provided moral support,
- the National Research Fund (NRF), for providing me with financial aid,
- Gert-Jan van Rooyen and Jaco de Witt for this thesis template and general help with various LaTeX issues,
- Herman Engelbrecht, for sharing his insights on many topics related (and sometimes not related) to this thesis,
- in alphabetical order: Eugene, George, Gert-Jan, Herman, Jaco and Willie, for enduring my rants, listening and commenting on my ideas, laughing at my jokes, and the many coffees shared during the last two years,
- Hansie, Lourens, Gid and Giep, for being such great friends, for all the laughs, the coffees, braais and general support,
- Nita, who always had time for a chat, especially non-work related and for being an inspirational friend,
- and lastly, Michelle, for all the love, support and patience the last eight years, for helping me keep my sanity, for always being interested in my work and for always believing in me.

Contents

List of Figures

List of Tables

Nomenclature

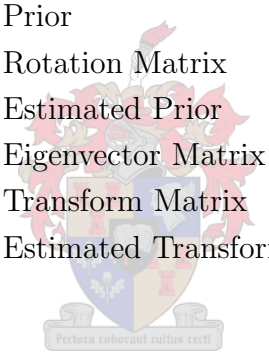
Acronyms

ASR	Automatic Speech Recognition
AST	African Speech Technology
CMS	Cepstral Mean Subtraction
DCT	Discrete Cosine Transform
EM	Expectation Maximisation
FA	Factor Analysis
GMM	Gaussian Mixture Model
HLDA	Heteroscedastic Linear Discriminant Analysis
HMM	Hidden Markov Model
KLT	Karhunen-Loéve Transform
LDA	Linear Discriminant Analysis
MFCC	Mel-scale Frequency Cepstral Coefficient
ML	Maximum Likelihood
ORED	Order Reducing
PCA	Principal Component Analysis
PDF	Probability Density Function
PPCA	Probabilistic Principal Component Analysis
PR	Pattern Recognition
T-BAGMM	Tree-Based Adaptive Gaussian Mixture Model

Variables

symbol	**description**
σ	Standard Deviation
σ^2	Variance
$\boldsymbol{\epsilon}$	Noise Element
ρ	Correlation Coefficient
$\boldsymbol{\mu}$	Mean Vector
$\hat{\boldsymbol{\mu}}$	Estimated Mean Vector

symbol	**description**
$\mathbf{\Sigma}$	Covariance Matrix
λ	Eigenvalue
$\boldsymbol{\lambda}$	Eigenvalue Matrix
$\hat{\mathbf{\Sigma}}$	Estimated Covariance Matrix
$\mathbf{\Psi}$	Noise Covariance Matrix
$\hat{\mathbf{\Psi}}$	Estimated Noise Covariance Matrix
$\mathrm{diag}(\cdot)$	Diagonal Operator
$p(\cdot)$	Probability
$\mathbf{B}$	Whitening Matrix
C	Number of Clusters/Classes
D	Dimension of set
$\mathrm{E}(\cdot)$	Expected Value
$\mathbf{I}$	Identity Matrix
M	Reduced Number of Dimensions
N	Number of samples in set
$P(\cdot)$	Prior
$\mathbf{R}$	Rotation Matrix
$\hat{P}(\cdot)$	Estimated Prior
$\mathbf{U}$	Eigenvector Matrix
$\mathbf{W}$	Transform Matrix
$\hat{\mathbf{W}}$	Estimated Transform Matrix



Chapter 1

Introduction

1.1 Motivation

Today Automatic Speech Recognition (ASR) systems are slowly gaining popularity. Some computer operating systems (such as *Microsoft Windows XP*) have these systems incorporated and they are also becoming popular in luxury cars. It is obvious that systems like these are very beneficial as extra input devices. When one is driving, it is much safer just to 'ask' the car to perform a task such as calling a number on the phone or selecting a radio station than doing it by hand and possibly losing control of the vehicle.

The technology driving these systems has not changed drastically in the last few years. While the full system implementations have been well documented [16, 23, 25], the finer details regarding the statistical models to calculate probabilities are not well documented and compared.

The motivation behind this thesis is not only to shed some light on the different statistical models (more specific all the Gaussian based models), but to do comparative tests between them and discuss the strong and weak points of each one. This includes classic approaches and also a few newly introduced ones, where implementation is also discussed.

When building models of speech utterances (phonemes in the case of this work), the statistical model usually consists of a mixture of smaller models. This ensures better coverage of the statistical properties of the segment of data we want to model. For the past few years, the prevalent base model for these mixture components has been the diagonal covariance Gaussian Probability Density Function (PDF) [23].

Traditionally three variants of multivariate Gaussian PDFs are used. The full covariance, diagonal covariance and spherical covariance Gaussian PDFs. These variants represent a range of precision. This ranges from a big generalisation (the spherical covariance) to no generalisation (the full covariance) of the Gaussian shape.

Practical results have shown (and we will again show this later in this thesis) that the spherical covariance matrix is a gross generalisation and is not practically usable in ASR systems of today. Full covariance Gaussians are very detailed, but very slow to train and to calculate the likelihoods. The only reasonable option is the diagonal covariance Gaussian,

as it is relatively fast to execute in both training and calculating likelihoods, and it retains enough information to be useful.

This means that currently the diagonal covariance Gaussian is the only realistic option to use in an ASR system. We are limited by the fact that to improve the mixture model, we can only raise the number of mixture components. This can introduce problems such as over training. If we want a more detailed model than the diagonal covariance, we are forced to use the full covariance model. This represents a huge difference in performance.

The motivation for this work is to find more intermediate options for the implementation of PDFs in an ASR system. These options should fill the void that exists between the diagonal and full covariance PDFs. Another useful addition is the description, implementation and benchmarking of such methods against their traditional counterparts.

1.2 Research Objectives

Although documentation regarding the methods discussed in this thesis exist in isolation, there is no one document encompassing all these methods together and comparing them to each other in terms of implementation. In this document we have the following objectives:

- Introducing three PDF classes from previous work [1, 8, 10, 14, 28] to be used in mixture models. These are the sparse-, the Probabilistic Principal Component Analysis- (PPCA) and the Factor Analysis (FA) covariance Gaussian models.
- Describing the mathematical description of these models.
- Showing examples of the practical usage of each model, including strengths and weaknesses.
- Providing comparative results for these and the traditional models on a working ASR system.

1.3 Concepts Relating to Statistical Modelling of Speech Data

When modelling speech data, the raw sound format is first processed into a form easy for recognition. We want to have the speech data in sets of equal length vectors to make modelling practical.

Preprocessing techniques are applied to the sound data to emphasise the higher frequencies and to make the speech segments all have equal average power [21]. Next we extract the feature vectors from the data as Mel-scale Frequency Cepstral Coefficients (MFCCs) using the Mel-scale filter bank and Discrete Cosine Transform (DCT) [3, 5, 21].

Next we do post-processing on the feature vectors (now consisting of cepstral coefficients). First we do Cepstral Mean Subtraction (CMS) to get rid of most of the recording channel

effects. We then normalise each dimension on a set of feature vectors to have unity variance. This is done in order to improve the calculation accuracy when using floating point numbers on a computer. Next we take each feature vector and append the preceding four and following four feature vectors to it. This includes the changes over time for the following features.

To decrease the size (or dimension) of the resulting feature vectors, we calculate a Linear Discriminant Analysis (LDA) transform. We take the parts of the speech we want to model under a single PDF and cluster them together into separate classes. The LDA not only reduces the dimensionality of the features, but also rotates them to ensure maximum separability between the classes we chose. The dimensions that have the highest separability between the classes are retained. An illustration of LDA is seen in figure 1.1.

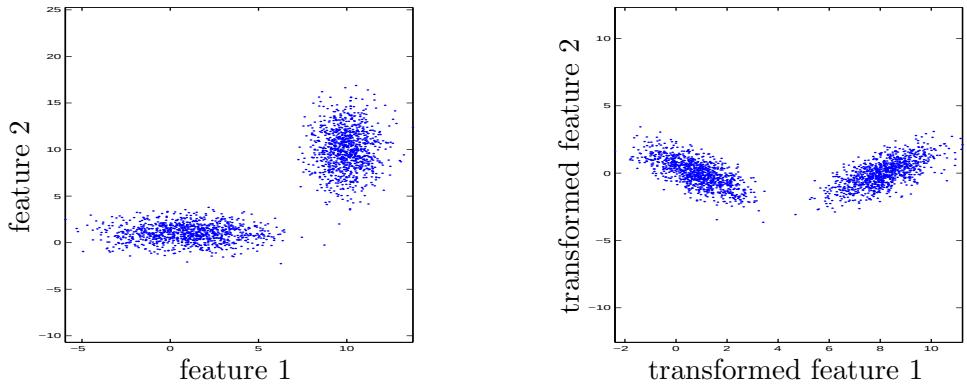


Figure 1.1: *A data set with two classes before and after LDA transformation.*

In figure 1.1 we see that before the LDA transform is applied, the two features contain similar information regarding class separability. After LDA transformation, the transformed feature 1 contains the maximum class separability information. Transformed feature 2 has no separability information (the two classes overlap each other fully) and can for the purpose of class separation be removed. This reduces the dimension from two to one while still retaining maximum class separation.

After these steps, we have the data in a form suitable to generate descriptive models. We model each phoneme by using a Hidden Markov Model (HMM). A three-state left-to-right HMM is shown in figure 1.2. Besides the start and stop states, this HMM contains three states that can be traversed from left to right. This creates a model that has three distinct time sections. Each state represents a certain time section of the sound. State one models the beginning of the phoneme, state two the middle and state three the end.

Each state of the HMM contains a Gaussian Mixture Model (GMM) to describe the features expected at this part of the phoneme. This mixture model consists of a sum of weighted Gaussian PDFs. When examining speech data (figure 1.3) we see the data is distributed in a non Gaussian way. It is therefor not practical to use one Gaussian PDF to calculate the statistics of the data. When using a GMM however, the data from figure 1.3

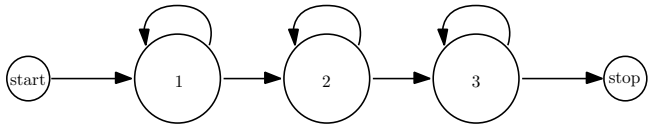


Figure 1.2: *A three-state left-to-right HMM.*

can be modelled in an accurate PDF (shown in figure 1.4).

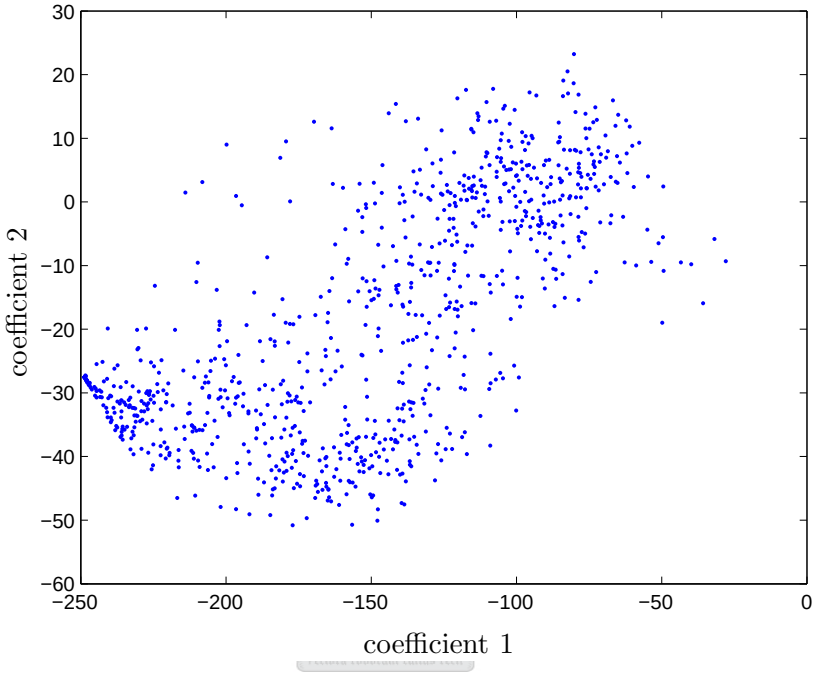


Figure 1.3: *A typical speech signal's first two cepstral coefficients.*

Using the above concepts we can model each phoneme with an HMM that describe certain time sections of the speech with GMMs.

1.4 Prior Work on PDFs

As mentioned before, the full, diagonal and spherical covariance Gaussian PDFs have been used for some time. Of these three, the diagonal covariance variant is the most used. Some work has been done to make up for the large gap between the full and diagonal covariance variants.

One of the ways to include more information is by the use of Heteroscedastic Linear Discriminant Analysis (HLDA) [12, 18]. This method would replace normal LDA in the post-processing of the speech signal. HLDA also considers the difference in variance for each class and applies this information for better class separation. After HLDA, models are

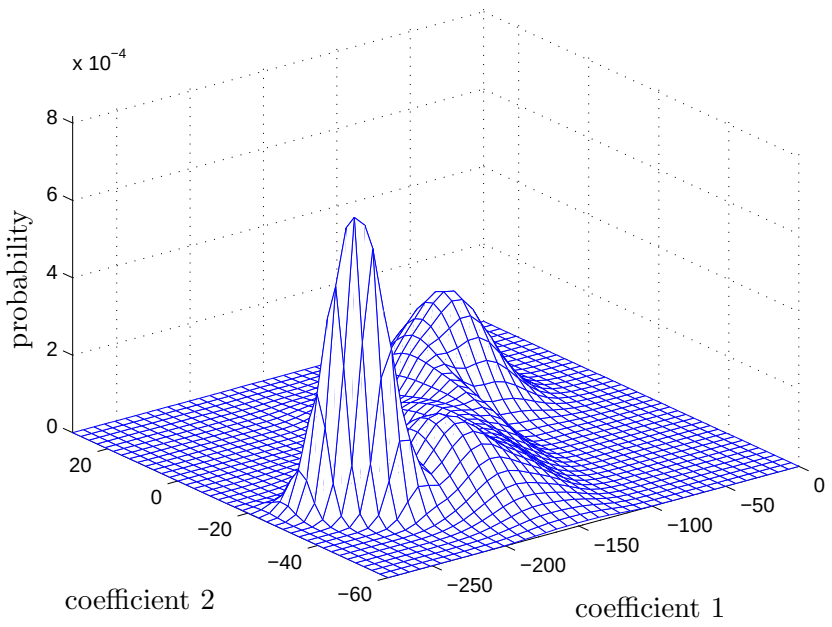


Figure 1.4: *A GMM representation of figure 1.3.*

estimated from the data in the same fashion as before. With HLDA, as with LDA, we do not improve the models, but we improve the data sent to the models.

The sparse covariance Gaussian PDF was developed for use in the Pattern Recognition system (PatrecII) of the Digital Signal Processing (DSP) group of the Stellenbosch University (SUN). It was conceived and implemented by Prof. Johan du Preez [8, 10].

This method was developed because there was a need to build better statistical models for pattern recognition than was possible with the diagonal covariance Gaussian PDF. This method also has the benefit of being able to scale to model more or less information as needed. This method has not been documented formally and will be discussed in depth in chapter 3.

The Probabilistic Principal Component Analysis (PPCA) covariance matrix is derived from the plain Principal Component Analysis (PCA) transform [1, 28]. With the PCA transform, one can extract the axes describing the most information. The PCA is commonly used for dimensionality reduction (see section 3.2).

While the PCA works well on a global scale, it is difficult to use it on individual mixture components of a data set as there is no correspondence between the PCA and a probability density structure. With the PPCA method, the PCA structure is reformulated in a maximum likelihood framework. The result of this and the previous work is a mathematical model for the PPCA method that can be used in a Gaussian mixture model (GMM) setup.

The PPCA covariance matrix can also be scaled to directly store a variable amount of information. The number of principal components explicitly modelled determine how much

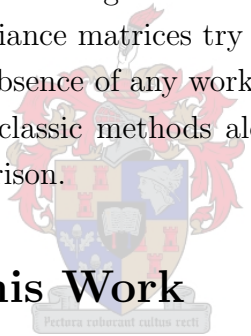
of the information is modelled.

Factor Analysis (FA) uses the same maximum likelihood framework of the PPCA method, but with a different assumption [1]. The PPCA method assumes the discarded information can be modelled isotropically (i.e. via a spherical covariance matrix). With the FA covariance matrix the discarded data is modelled via a diagonal covariance model. Using the same structure as the PPCA method means it is equally usable in GMMs. One of the disadvantages of the FA computation is that unlike the PPCA method there is no closed loop solution and the parameters need to be estimated via a Expectation Maximisation (EM) algorithm implementation. An algorithm for simultaneously training the EM parameters of the GMM and FA covariance mixture components is proposed in the literature [14].

The FA covariance matrix can be scaled in a similar manner as the PPCA covariance matrix. In this case the number of factor loadings to compute determines the complexity of the model.

It is clear that there have been a few attempts to formulate new processing methods and PDFs to make up for the gap between the diagonal and full covariance matrices. One way is to improve the information in the data by the HLDA. Another is to retain more information from the covariance matrix and discarding information that is less important. This is what the sparse-, PPCA- and FA covariance matrices try to achieve.

Another thing to note is the absence of any work to directly compare all these methods. Even a comparison between the classic methods alone is scarce. One of the aims of this thesis is to provide such a comparison.



1.5 Overview of This Work

This section contains a summary of the work done in this thesis. This summary contains discussions on the methods implemented and issues surrounding them. Motivation and detail are left to the chapters describing these methods and their implementation. Chapter 2 describes the theory, implementation and properties of the three traditional Gaussian PDFs, namely spherical, diagonal and full covariance. Chapter 3 does the same for the three new PDFs: the sparse-, PPCA- and FA covariance Gaussians. Chapter 4 describes the test system used for evaluating these methods and chapter 5 the results of these evaluations.

The following sections will outline the work in these chapters:

1.5.1 The Classic Gaussian PDF variants

The three classic PDFs used in GMMs are the full, diagonal and spherical covariance Gaussian PDFs. As mentioned before, the diagonal is the method used most often, as it is fast and gives reasonably good models of the data.

Al three these methods are very easy to implement. The full covariance Gaussian requires estimating the covariance from the data, a well established statistical procedure.

Determining the diagonal covariance Gaussian is just as easy, as one discards all the off diagonal components of the full covariance matrix. One can also take advantage of the fact that only the diagonal of the matrix is wanted and adjust the equations to estimate only the diagonal.

Estimating the spherical covariance matrix is achieved by taking the mean of the elements on the diagonal of the covariance matrix. Again the math can be simplified by only estimating this value.

These three methods are all used in a GMM by using the Expectation Maximisation (EM) algorithm [20]. The first (Expectation) step requires calculating the likelihood of each feature for each mixture component. The calculation of the likelihood depends on the type of PDF used for the component. Calculating the likelihood for the simpler methods, like the diagonal and spherical covariance Gaussians, is faster than for the full covariance Gaussian.

The second (Maximisation) step involves using the information of step one to regroup feature data into the mixture components and reestimating the component statistics. Estimation of the statistics depends again on the type of mixture component used. These two steps are repeated until the estimation converges.

The models estimated by these methods also differ. The full covariance Gaussian is the best model, as it not only models the covariance of each dimension, but also all of the correlations between dimensions. It needs a lot of training data to estimate a reasonable model and less mixture components are needed than with other PDFs. It is very slow to estimate and also slow to calculate the likelihood.

The diagonal covariance Gaussian only models the variance of each dimension. Correlations between dimensions are ignored. It needs less training data than the full covariance Gaussian, but more mixture components are needed to model data efficiently. It is fast to estimate and also fast to calculate the likelihood. The gain in time over calculation overshadows the loss in time caused by using more mixture components. This makes it more practical to use than the full covariance Gaussian.

The spherical covariance Gaussian models the average covariance of all the dimensions and ignores individual covariances and correlations. It needs very little training data, but to use in a mixture model many mixture components need to be used. It is very fast to estimate and calculate the likelihood. This model is really only practical when data is very scarce. The overhead in calculating the EM-algorithm is pronounced because so many mixture components are needed. This method is therefor only practical in very special circumstances.

When using these methods, the dimensionality of the feature vectors usually have been reduced via the LDA before modelling takes place (later discussed in section 3.2). After the LDA transform, the average covariance matrix of all the predetermined data clusters is the identity matrix. Each of these clusters will be individually modelled by a GMM.

With the average cluster covariance matrix being the identity matrix, it means that the data in each cluster has been scaled in such a way that the covariance matrix is as close to

identity as possible. This means that correlations between dimensions are at a minimum. This makes the assumption of the diagonal covariance matrix more feasible.

With the full covariance Gaussian, a lot of computation goes into the correlations between the dimensions. When these values are minimised, it makes more sense to use the diagonal covariance Gaussian.

The LDA transform also favours the spherical covariance Gaussian, because the variances of the dimensions are closer to each other than before the LDA.

Realistically, in speech data these clusters are not similarly shaped. Even if the average covariance matrix is the identity matrix, the individual cluster covariance matrices could still be far from identity. The LDA does, however, scale these matrices closer to the ideal than before the transform.

1.5.2 The New Gaussian PDF variants

The three new Gaussian PDFs in this thesis are the sparse-, PPCA- and FA covariance Gaussian PDFs. The aim of this work is to examine the implementation and performance of each of these methods in relation to the classic methods mentioned above.

All three methods have a trickier implementation than the classic methods. They all are more complex than the diagonal covariance Gaussian, but they all fill the space between the diagonal and full Gaussians. Another benefit is the scalability of these methods. They can be made progressively more complex up to the same complexity as the full Gaussian.

Estimating all these equations does require the estimation of a full Gaussian covariance matrix first. With the sparse covariance Gaussian we calculate the correlation coefficient matrix from this covariance matrix. The minimum correlation coefficient value is then used to discard any correlations below this threshold. The correlation coefficients are then evaluated from the largest to the smallest. Each value represents two dimensions that are correlated. All correlated dimensions are stored in blocks. The length of each block is limited by the maximum block length parameter. When the blocks have been calculated, only the covariance values corresponding to these blocks are stored. When calculating the likelihood only these blocks and the diagonal are considered.

When using the sparse covariance Gaussian, the model has full covariance within the correlation blocks, but diagonal covariance between them. If the dimensions were to be rearranged so that the correlated dimensions were adjacent, the resulting covariance matrix would only have had blocks of non-zero values around the main diagonal. This allows for more precision between dimensions that are heavily correlated.

The sparse covariance Gaussian needs more training data than the diagonal, but the amount of data needed can be determined by the size of the blocks. When using this model in a GMM, the number of mixture components needed for a good result therefor depends on the maximum size for these correlation blocks. Estimating the models and calculating the likelihood do take longer than for only the diagonal, but the modelling accuracy is also improved.

The estimation of the PPCA covariance Gaussian requires finding the principal components of the covariance matrix. The number of components to consider is determined by a preset parameter and also by the relative value of the eigenvalue corresponding to each principal component. The discarded components are generalised in a spherical covariance Gaussian. The sum of the information contained in the principal components and the spherical covariance element combine to give the PPCA covariance Gaussian representation. When the likelihood is calculated, only these values are used for the calculation.

The PPCA covariance Gaussian models the directions of the principal components fully. In these directions all the information is retained. The remaining directions orthogonal to the principal directions are generalised to one value (as the whole covariance matrix is represented by the spherical covariance Gaussian).

This method also needs more training data than the diagonal covariance method. With every principal component, the required amount of data increases. In a GMM, the number of mixture components for a good representation thus depends on the number of retained principal axes. This method also takes longer to evaluate or estimate than the diagonal does, because it uses more information.

The estimation of the FA covariance Gaussian relies on finding the factor loadings of the covariance matrix via factor analysis. This method is not closed loop and requires a few iterations to converge. The number of factor loadings to calculate is predetermined. The rest of the covariance information is generalised into a diagonal covariance matrix. The sum of the information in the diagonal element and the retained factors gives the combined FA covariance Gaussian representation. When the likelihood is determined, only these parameters are considered.

The FA covariance Gaussian explicitly models the information in the factor loadings. As with the PPCA method we have some directions that are modelled completely. The remaining information is modelled by a diagonal matrix, such as the diagonal covariance Gaussian does. This means the directions orthogonal to the ones corresponding to the factor loadings are modelled, but the correlations between them are not.

This method needs more training data than the diagonal covariance Gaussian, but the amount of data needed is determined by the number of factor loadings. In a GMM, the number of mixture components needed for a good representation therefor relies on the number of factor loadings per component. This method takes longer to estimate than the diagonal- and PPCA covariance Gaussians. It also takes longer to calculate the likelihood than the diagonal does because more information is modelled.

These three methods can be used in a GMM in a similar fashion as the classic methods. The EM-algorithm is implemented in the same manner. With the expectation step, the likelihoods are calculated according to each method. In the maximisation step, the re-estimation is done for the covariance matrix of each mixture component depending on which PDF is used.

With the FA covariance Gaussian one would have to do an iterative re-estimation for

each mixture component, for each step of the EM-algorithm. This would be very time consuming. To remedy this, only one iteration of the re-estimation calculation is done for each component, using the previous values of the components as the initial conditions. This proved to converge simultaneously with the EM-algorithm for calculating the GMM.

1.5.3 Implementation of Test System

To accurately evaluate the above PDFs, it is necessary to implement them in a real world ASR system. For the purpose of this thesis, two sets of speech corpora were used. The first is the NTIMIT speech corpus (see Appendix A for more detail on speech data sets). This corpus has time aligned transcriptions for each utterance. This makes it possible to identify the segments of each utterance that contain the phoneme data.

These segments are identified and used to create phoneme HMMs (similar to the one in figure 1.2) that model each phoneme. These HMMs are then used to evaluate a test data set (not used in training). The phonemes in this test set are evaluated and automatically transcribed. These transcriptions are then compared to the test set's (assumed) correct transcriptions. An accuracy score is calculated based on correct placings and wrong phoneme insertions. These scores are used to evaluate each PDF type used as base for the GMMs that are in the HMM states.

The second data set is the AST speech corpus. This corpus does have transcriptions for each utterance, but these transcriptions are not time aligned. A different technique, called embedded training, is used to train the phoneme HMMs for this system. This technique needs an initial set of phoneme HMMs. We use the set from the previous NTIMIT corpus to initialise this training system.

The HMM phonemes are trained with each of the six PDF types investigated in this thesis. The diagonal covariance Gaussian is used for the baseline systems, as it is the standard base PDF used in ASR.

For the sparse covariance Gaussian we choose to train with a maximum block size of two and a minimum correlation coefficient value of 0.4. This results in a model very close to the diagonal covariance Gaussian. The time difference in evaluation is therefor only due to a few extra parameters.

For the PPCA covariance Gaussian we choose to train with one principal component and we limit the eigenvalue of the evaluated components to 0.083 (see Chapter 3). Similarly we train an FA covariance Gaussian system with one factor loading. These are again one iteration up from the diagonal for each model.

In doing these tests it is found that the three new PDFs do indeed model more information than the diagonal covariance Gaussian. They all give better results with a slower evaluation time than the diagonal covariance Gaussian. The models also have more free parameters, confirming the fact that they do store more information than the diagonal covariance Gaussian.

Although the three newly implemented PDFs were more accurate, slower and larger

than the diagonal covariance Gaussian, they were less accurate, faster and smaller than a full covariance Gaussian. This confirms that these methods do indeed fill the gap between these two.

We next trained a system with each method to match the baseline system accuracy by altering the number of mixture components. These systems are evaluated and the evaluation times are compared.

The result of this test showed that the three newly implemented PDFs have slower evaluation times than the diagonal covariance Gaussian at the same accuracy. This further encourages the use of diagonal covariance above any other method. Retrained systems on a data scarce version of the AST speech corpus showed that while there was some improvement in the number of mixture components needed in some cases, the diagonal covariance system still was the fastest for the given accuracy.

1.6 Contributions

1. Although the classic distribution models, namely the full, diagonal and spherical covariance Gaussian PDFs are well known, not one text could be found that directly compare these methods to each other and comment on their strengths and weaknesses:

   (a) Implementation of these three methods is very simple. No complicated math is needed, as simple assumptions on the full covariance Gaussian lead to the diagonal and spherical covariance Gaussians.

   (b) Visual examples give the reader a more intuitive idea how these methods model the data. Full covariance Gaussians model each dimension and all correlations. The diagonal covariance Gaussian only models the variance of each dimension and the spherical covariance Gaussian models the average covariance of all the dimensions.

   (c) The comparison of these three methods confirms the popularity of the diagonal covariance Gaussian. Although the full covariance Gaussian is found to be the most accurate, it is slower and much larger than the other methods. The spherical covariance Gaussian is very inaccurate and not practically usable. The diagonal covariance Gaussian is fast and accurate enough to model the data effectively.

2. Three additional methods are considered for the use in ASR systems. The sparse-, PPCA- and FA covariance Gaussian PDFs are specifically created to fill the void between the diagonal and full covariance Gaussians. Not only are these implemented, but compared to each other and the classic PDFs:

   (a) There are some issues regarding implementation by these three methods. These issues are discussed and the full implementations are described. Matlab code for these implementations are supplied in Appendix B. The algorithm for calculating

the sparse covariance matrix is not trivial and requires a few passes through the covariance matrix elements. The PPCA covariance matrix elements only require an eigenvalue decomposition and some minor calculations. The FA covariance matrix requires factor analysis, which is an iterative process.

(b) Some visual examples give the reader a sense of what extra information is regarded important to model for these three methods. The sparse covariance Gaussian acts in some dimensions like a full covariance Gaussian and in others like a diagonal covariance Gaussian. Which correlations to model is determined by the input parameters. With the PPCA covariance Gaussian the principal directions are modelled fully, while the remaining data is generalised by a spherical covariance matrix. The FA covariance Gaussian also fully models the information in the factor loadings, but generalises the remaining information with a diagonal covariance matrix.

(c) When comparing these models to the classic methods, we find that all three improve in accuracy on the diagonal covariance Gaussian. They are still faster and smaller than the full covariance Gaussian. These methods do fit in the gap between the full and diagonal covariance Gaussians. All three have differing accuracy and speed depending on their input parameters. This is due to the fact that each one models extra information according to their own measures. It is difficult to directly compare these methods as their scale factors are not equal.

3. Evaluation of these methods in a real world ASR system is needed to fully examine the differences. A test system is implemented and a few tests are run:

(a) Implementation of a phoneme recogniser is discussed broadly. Two datasets are used, with one having transcriptions that are time-aligned (the exact time of each word occurrence is known) and one dataset that has transcriptions where the exact time occurrence of each word is not known. Issues regarding signal processing, pattern recognition and testing are discussed. With time-aligned transcriptions the phonemes are easy to identify and extract from the speech data. With non-time-aligned transcriptions a process called embedded training is used to train the phoneme models since it is impossible to establish which feature vectors belong to which phoneme (or phoneme section).

(b) The results of the comparison between the PDFs are shown and discussed. The error rate gains/losses are shown in table 1.1. Again it is seen that the diagonal covariance Gaussian gives the best balance between accuracy, speed and size when comparing the classic models. In our test system, it has a relative 18% higher error rate than the full Gaussian, but is 355% faster. The spherical covariance Gaussian has a 14% higher error rate than the diagonal covariance Gaussian and is 30.6% faster. It was found though, that even when doubling the mixture components,

it still does not come close to the diagonal covariance Gaussian in accuracy and becomes progressively slower.

(c) When comparing the newly implemented models it is found that the three models improve the accuracy over the diagonal covariance Gaussian by 0.3–4% (see table 1.1). Evaluation took from between 34–77% longer. It is clear that a gain in accuracy is achievable with penalties in speed. The sparse covariance and PPCA Gaussians have added functionality, because an extra parameter is used to evaluate if correlations are strong enough to model, making it faster and smaller on data that is not heavily correlated. The FA covariance Gaussian explicitly models data, regardless of the strength of correlations. The FA covariance Gaussian makes up for this fact by modelling the discarded dimensions more accurately with a small speed penalty over the PPCA method, ensuring higher accuracies.

PDF (Gaussian) Type	Gain(+)/Loss(-)	
	Error Rate	Speed
Spherical	+13.7%	+30.6%
Full	-18.4%	-355.8%
Sparse	-1.4%	-34.8%
PPCA	-0.3%	-69.6%
FA	-4%	-77.9%

Table 1.1: *Resultant gains/losses in error rate and speed over the diagonal covariance Gaussian PDF for each PDF type with equal mixture components.*

(d) The results of further comparisons are shown. Mixtures for each method are scaled to get close to the accuracy of the diagonal covariance Gaussian. The speed of each method is then compared. Additional tests are run with limited mixture components and limited training data. The general results are shown in table 1.2 (note that the spherical covariance Gaussian could only yield comparative results with lower mixture components). The results on table 1.2 shows that none of the methods manage to beat the diagonal covariance Gaussian. This implies that the diagonal covariance Gaussian is still the best method to use and taking more mixture components is more efficient than trying to model the correlations between dimensions.

PDF (Gaussian) Type	Gain(+)/Loss(-) Speed
Spherical	-23.5%
Full	-(142.9–200.5)%
Sparse	-(28.9–53.3)%
PPCA	-(68.7–82.2)%
FA	-(53.7–61.9)%

Table 1.2: *Resultant gains/losses in speed over the diagonal covariance Gaussian PDF for each PDF type with scaled mixture components for similar accuracy (ranged between various testing systems).*

Chapter 2

The Classic Distribution Models

2.1 Introduction

In this chapter we focus on the classic PDFs used to model speech data. These models are mostly used in GMMs to approximate complex data. In the case of phoneme recognisers, the GMMs model each of the states of an HMM. The models discussed in this chapter are:

1. the full covariance Gaussian distribution model,
2. the diagonal covariance Gaussian distribution model, and
3. the spherical covariance Gaussian distribution model.

For each one we begin by describing the theory behind the model. This includes mathematical formulae. We also discuss, if relevant, the reason for the development of the models. Implementation and the issues thereof are also discussed, followed by a discussion of the strengths and weaknesses of each model.

For illustrative purposes we select a common test dataset to illustrate the behaviour of each model. This set can be seen in figures 2.1 and 2.2. It is a 3 dimensional spiral that has the following properties:

- On the X-Y plane it has the shape of an annulus that is evenly distributed between the radius of 3 and 4.
- On the Z plane it has a constant increasing slope with a Gaussian noise element added.

We use the EM algorithm [20] to train GMMs with each PDF type and examine how they fit this test set. We see how many mixture components are needed for a good representation and compare the differences between each PDF.

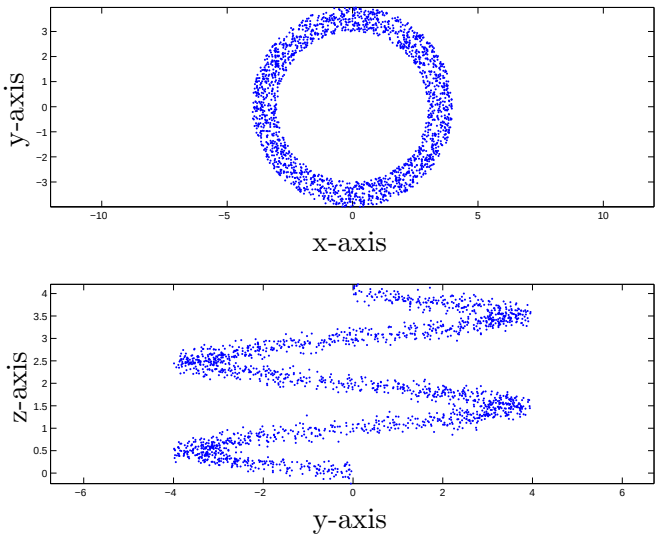


Figure 2.1: *A test dataset for illustrative purposes (2 dimensional view).*

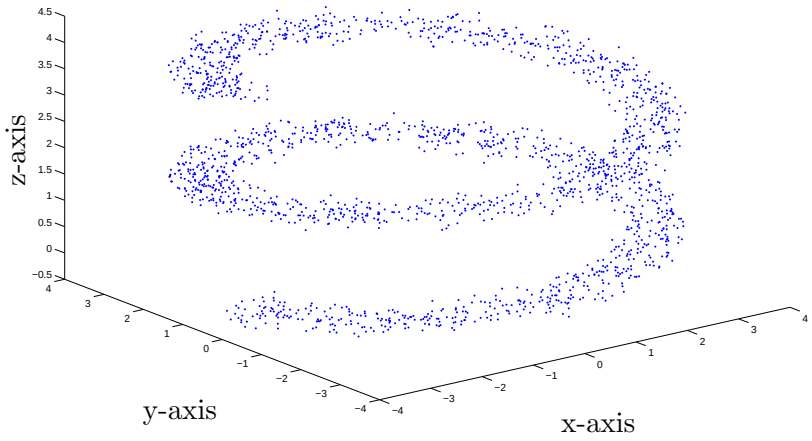


Figure 2.2: *A test dataset for illustrative purposes (3 dimensional view).*

2.2 The Full Covariance Gaussian PDF

2.2.1 Theory

The full covariance Gaussian PDF is the base for all the other models discussed in this thesis. It is the model that holds the most information. It gives all the statistics over a certain data collection with the assumption that the data is normally distributed.

Let us look at the one dimensional case. The Gaussian PDF is mathematically formulated as

$$p(x) = \frac{1}{\sqrt{2\pi\sigma_x^2}} e^{-(x-a_x)^2/2\sigma_x^2} \tag{2.1}$$

where $\sigma_x > 0$, $-\infty < a_x < \infty$ and $p(x)$ is the density of x. In figure 2.3 we can see Gaussian representations for a_x equal to -1, 0 and 1. It illustrates that a_x denotes the mean value

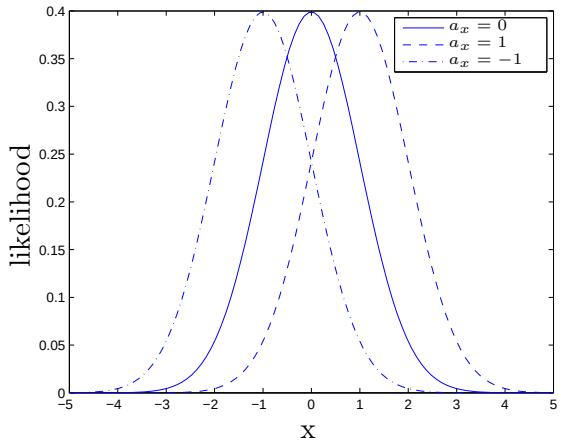


Figure 2.3: *Single dimension Gaussian with varying mean values.*

of the PDF, or in simpler terms it gives us the value of x around which the distribution is centred. It can also be said that a_x is the expected value of the distribution. This is evident as it has the highest likelihood.

In figure 2.4 we illustrate Gaussian representations with varying values for σ_x. This changes not only the 'spread' of the function, but also the size of the peak. The latter is because a true PDF always has an area of one. We call σ_x the standard deviation of the function and σ_x^2 is defined as the variance.

The maximum value for the Gaussian PDF is $(2\pi\sigma_x^2)^{-1/2}$. The function has 0.607 times its maximum value at the points $x = a_x \pm \sigma_x$.

The Gaussian PDF is popular because it models the density function of many real life events that are deemed to be random. The Gaussian is also a simple statistical model as all the moments can be calculated as functions of the mean and variance and these are the only values needed to compute the Gaussian approximation of an event [11].

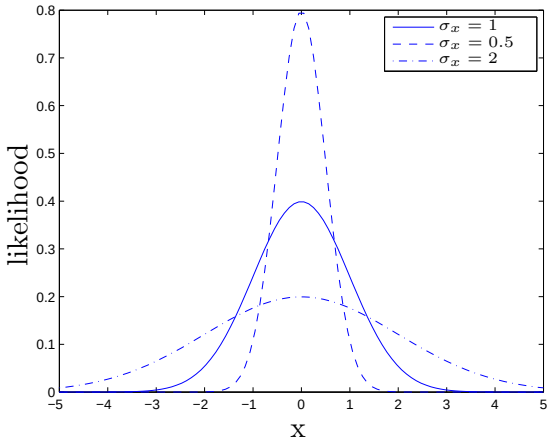


Figure 2.4: *Single dimension Gaussian with varying standard deviation values.*

In general, we are usually working with more than one dimension in a PR problem. Thus we are working with the multivariate Gaussian PDF. The multivariate Gaussian PDF has at its input a multi dimensional vector. It has a mean vector $\boldsymbol{\mu}$, instead of a mean value, that has a mean value for each dimension. The variance is also replaced by $\boldsymbol{\Sigma}$, the covariance matrix. The covariance matrix is a square matrix that has as many rows and columns as there are dimensions. Each row and each column represents a certain dimension. The elements of $\boldsymbol{\Sigma}$ are a function of the variance and correlation between the dimensions belonging to the specific row and column. On the diagonal we have the variances of each dimension. This gives as the following expression for the multivariate Gaussian PDF:

$$p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\} \tag{2.2}$$

where $p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ reads "the density of $\mathbf{x}$ given $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$", and D is the number of dimensions [27]. The mean vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ will be in the form:

$$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \vdots \\ \mu_{D-1} \\ \mu_D \end{bmatrix}, \qquad \boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & C_{12} & C_{13} & \cdots & C_{1(D-1)} & C_{1D} \\ C_{21} & \sigma_2^2 & C_{23} & \cdots & C_{2(D-1)} & C_{2D} \\ C_{31} & C_{32} & \sigma_3^2 & \cdots & C_{3(D-1)} & C_{3D} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ C_{(D-1)1} & C_{(D-1)2} & C_{(D-1)3} & \cdots & \sigma_{(D-1)}^2 & C_{(D-1)D} \\ C_{D1} & C_{D2} & C_{D3} & \cdots & C_{D(D-1)} & \sigma_D^2 \end{bmatrix}$$

where $C_{xy} = \rho_{xy}\sigma_x\sigma_y$ and ρ_{xy} is the correlation coefficient between x and y.

Now, if we have a set of points that we assume are Gaussian, and we have a sufficient number of points, we can approximate the first and second moments (mean and variance) by

using the *law of large numbers* [22]. The equations for the above statistics are the following:

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \tag{2.3}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N-1}\sum_{n=1}^{N}(\mathbf{x}_n - \hat{\boldsymbol{\mu}})(\mathbf{x}_n - \hat{\boldsymbol{\mu}})^T \tag{2.4}$$

where N is the number of samples in the data.

The data used for $\mathbf{x}$ above is known as the *training* data. $\hat{\boldsymbol{\mu}}$ is approximated by the mean of the training data and $\hat{\boldsymbol{\Sigma}}$ is approximated using the mean subtracted training data.

Now we can take a set of test vectors $\mathbf{y}$ and substitute the above $\hat{\boldsymbol{\mu}}$ and $\hat{\boldsymbol{\Sigma}}$ in equation 2.2. We can then calculate the likelihood of $\mathbf{y}$ belonging to the same set as the training data $\mathbf{x}$. This is called *scoring*. Generally when we want to score a test set, we use *log likelihoods* instead of probabilities. The log likelihood is the log value of the likelihood. This simplifies the math by turning multiplication into addition and division into subtraction. When log base e is used, we can also discard the exponent operation [29]. The resultant equation for the log likelihood is:

$$\log_e(p(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma})) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) - \frac{D}{2}\log_e(2\pi) - \frac{1}{2}\log_e(|\boldsymbol{\Sigma}|) \tag{2.5}$$

where $\mathbf{x}$ represents the test vector.

2.2.2 Implementation

Implementing the full covariance Gaussian PDF is as simple as implementing equations 2.3–2.5. For training we only need one iteration as equations 2.3 and 2.4 have a closed loop solution. Thus estimating the mean and covariance matrix is sufficient for a full model of the Gaussian.

Now if we have a series of vectors and we want to know the chance that these vectors were indeed generated by the estimated model, we use equation 2.5 to get the likelihood. When we have a series of vectors, the likelihood of all of them originating from one model is their individual likelihoods multiplied with each other. In the log case we just add all the likelihoods. If we want to translate the final answer back to probabilities we just get the exponent. The result will probably be a very small number. This is another reason why we work with log likelihoods, as it prevents numerical underflow.

The answer for the above may not mean much numerically. To interpret the results you need to compare results. Usually one has many models and want to know which one is more likely to generate the given set of points. Comparing results, the one with the highest numerical value would be the one most likely to generate the data (hence the term *likelihood*).

Next we want to implement a GMM with the full covariance Gaussian as base. The GMM is trained with the EM algorithm. The GMM is a set of Gaussian PDFs, added together to form one PDF. Each component is weighted by a prior probability and all the

prior probabilities add up to one (to keep the volume of the density equal to one). A representation of a GMM of figure 1.3 is shown in figure 1.4 on page 5. We can see that a decidedly non-Gaussian distribution is effectively modelled by the GMM.

The EM algorithm uses Maximum Likelihood (ML) estimation to find the parameters of the GMM. The first decision is how many mixture components to use. The components are then initialised (by binary split and/or K-means for best results [27]). Iterative training then takes place. The current parameters are used to calculate the probabilities for each component (E-step). Then we take the results of the E-step and recalculate the parameters (M-step) using ML estimation [20]. The two steps are repeated until the parameters of the GMM converge.

In this case, the E-step entails the following:

$$P(k|\mathbf{x}_n) = \frac{p(\mathbf{x}_n|k)P(k)}{p(\mathbf{x}_n)} \tag{2.6}$$

where $p(\mathbf{x}_n|k)$ is the likelihood of $\mathbf{x}_n$ being generated by mixture component k, $P(k)$ is the prior probability of component k and

$$p(\mathbf{x}_n) = \sum_{k=1}^{K} p(\mathbf{x}_n|k)P(k) \tag{2.7}$$

where equation 2.7 is the likelihood of $\mathbf{x}_n$ being generated by the GMM.

In the M-step we calculate the following:

$$\hat{P}(k) = \frac{1}{N}\sum_{n=1}^{N} P(k|\mathbf{x}_n) \tag{2.8}$$

$$\hat{\boldsymbol{\mu}}_k = \frac{\sum_{n=1}^{N} P(k|\mathbf{x}_n)\mathbf{x}_n}{\sum_{n=1}^{N} P(k|\mathbf{x}_n)} \tag{2.9}$$

$$\hat{\boldsymbol{\Sigma}}_k = \frac{\sum_{n=1}^{N} P(k|\mathbf{x}_n)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k)^T}{\sum_{n=1}^{N} P(k|\mathbf{x}_n)} \tag{2.10}$$

We repeat for equations 2.6 to 2.10 until the sum of 2.7 for all values of $\mathbf{x}_n$ converges.

2.2.3 Strengths and Weaknesses

First we train a GMM to fit the data in figure 2.1. We find with repeated experiments that twelve mixture components give a good fit, as can be seen in figures 2.5 and 2.6. Note that the ellipses plotted in these figures are only an indication of the variance shape of the mixture components and do not indicate the full spread of the components. The prior probabilities are also not included in these figures. We only illustrate the position (mean) and shape (variance) of each mixture component.

With twelve mixture components the full covariance GMM is a good approximation of the dataset. If we now want to score a test set to this PDF, each feature needs to score against twelve full covariance Gaussians. In this lies the weakness of the full covariance

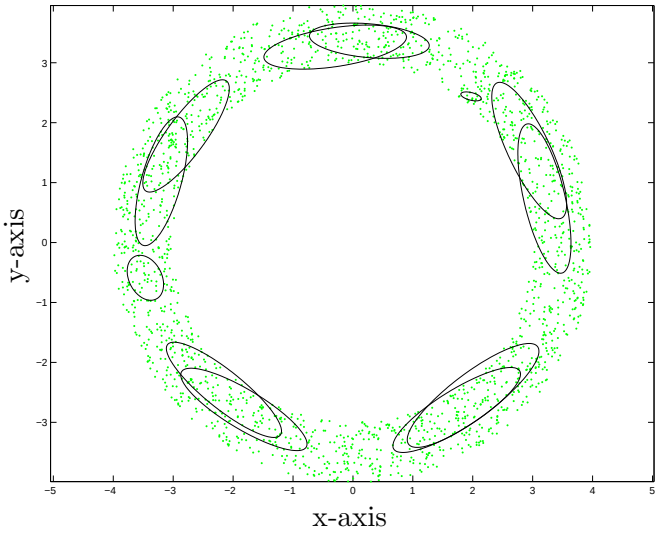


Figure 2.5: *Top view of a full covariance GMM fit to test set (12 mixture components).*

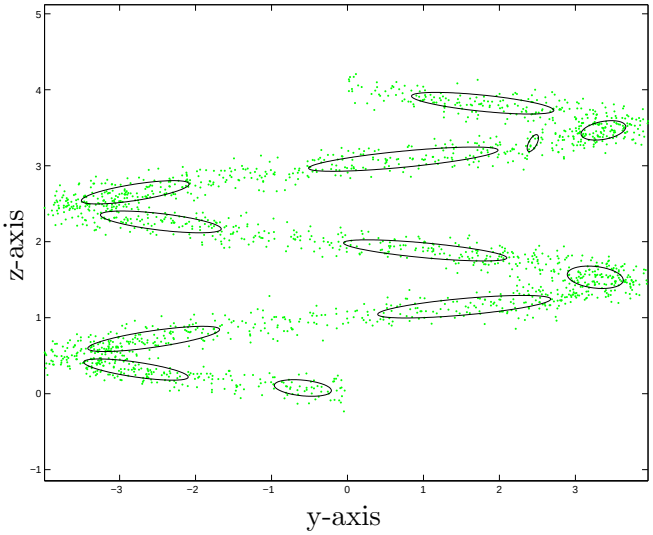


Figure 2.6: *Side view of a full covariance GMM fit to test set (12 mixture components).*

Gaussian. It is mathematically very expensive. With the training of the GMM, each time one needs make as many multiplications as there are terms in the full matrix. When the training is completed the final estimated covariance matrix is inverted and stored along with its determinant (see equation 2.5)

Another weakness is the fact that a full matrix needs to be inverted. This means the matrix needs to be of full rank [19]. At least as many linear independent features are needed as the number of dimensions. For large dimensional systems (as speech models typically are) one thus needs more training data than other methods. To get a good estimation, full covariance Gaussians require even more data per mixture component. This implies full covariance Gaussians usually do not fare well with data scarce systems.

The strength of the full covariance Gaussian is its precision. By using a full matrix for each mixture component, maximum information is stored for each component. This means it uses fewer components to model a dataset than the other methods in this thesis, however, one must beware of the danger of over training. If one adds more components, the model complexity increases substantially. This can be seen in figure 2.5 where there is a small Gaussian at the top right. This is a mixture component that modelled part of the dataset that is only slightly more dense than the surrounding parts. This shows that the full GMM is not easily scalable via mixture component adjustment.

Another strength of the this method is the simplicity of implementation. Although the algorithms used are expensive, they are relatively easy to implement.

If time and storage space are no issue, this method would be suitable.

2.3 The Diagonal Covariance Gaussian PDF

2.3.1 Theory

The diagonal covariance Gaussian PDF is a simplification of the full covariance Gaussian PDF. It makes the simple assumption that there are no correlations between dimensions, in other words it assumes that all the dimensions are statistically independent. We take equation 2.2 and redefine $\mathbf{\Sigma}$ to be the following:

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1^2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2^2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \sigma_3^2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \sigma_{D-1}^2 & 0 \\ 0 & 0 & 0 & \cdots & 0 & \sigma_D^2 \end{bmatrix}$$

The above matrix has a lot of entries that are zero. It would make more sense to save the diagonal values of $\mathbf{\Sigma}$ in a vector rather than a matrix.

Estimating the statistics from the training data would only change the way the covariance is estimated. The mean estimation (equation 2.3) would stay the same and equation 2.4

would be replaced by:

$$\hat{\sigma}_i^2 = \frac{1}{N-1}\sum_{n=1}^{N}(x_{in} - \hat{\mu}_i)^2 \tag{2.11}$$

with $\hat{\sigma}_i^2$ being the estimation of the ith diagonal element, x_{in} is the ith element of $\mathbf{x}_n$ and $\hat{\mu}_i$ the ith element of $\hat{\boldsymbol{\mu}}$. With equations 2.3 and 2.11 the log likelihood becomes:

$$\log_e(p(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\sigma}^2)) = -\frac{1}{2}\sum_{i=1}^{D}\frac{(x_i - \mu_i)^2}{\sigma_i^2} - \frac{D}{2}\log_e(2\pi) - \frac{1}{2}\sum_{i=1}^{D}\log_e(\sigma_i^2) \tag{2.12}$$

2.3.2 Implementation

As with the full covariance Gaussian, the implementation of the diagonal model is as easy as implementing equations 2.3, 2.11 and 2.12. This method also has a closed loop solution. As long as the training data does not change, only one iteration of these equations is needed for estimation.

As previously we score the diagonal covariance Gaussian with the likelihood in equation 2.12. The likelihood is interpreted in the same way as with the full covariance Gaussian. It is intuitive that the diagonal is a simpler model than the full model. More on this in section 2.3.3.

The same issues regarding interpreting the log likelihood discussed in section 2.2.2 are true for the diagonal covariance Gaussian: likelihoods for a vector can be summed for the total likelihood and models are scored and compared in exactly the same way.

Implementing the GMM with the diagonal covariance Gaussian as basis is very similar than with the full covariance GMM. The EM algorithm is still used and in the E-step all that change is the way that $p(\mathbf{x}_n|k)$ is calculated.

With the M-step, all remains the same, except the calculation of the covariance matrix, which now becomes:

$$\hat{\sigma}_{ik}^2 = \frac{\sum_{n=1}^{N} P(k|\mathbf{x}_n)(x_{in} - \hat{\mu}_{ik})^2}{\sum_{n=1}^{N} P(k|\mathbf{x}_n)}, \qquad i = 1, \ldots, D \tag{2.13}$$

Again we repeat the EM iteration until convergence.

2.3.3 Strengths and Weaknesses

We train a GMM with diagonal covariance Gaussians to fit the dataset in figure 2.1. First we take twelve mixture components, the same number that gave a good model with full covariance Gaussians. The results can be seen in figures 2.7 and 2.8.

We can see that the diagonal covariance GMM does not create such a good model for the data as the full GMM did. This is because none of the correlations between dimensions are modelled by the diagonal covariance Gaussians. One can see this in the fact that the ellipses in figures 2.7 and 2.8 only models information in the directions of the axes (in other

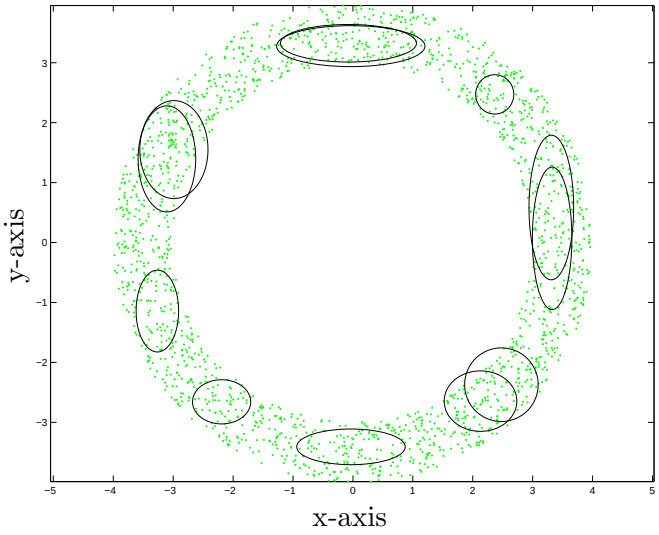


Figure 2.7: *Top view of a diagonal covariance GMM fit to test set (12 mixture components).*

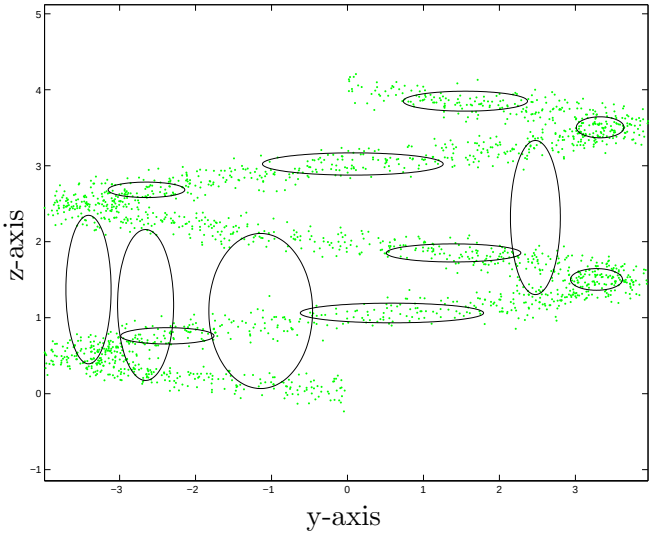


Figure 2.8: *Side view of a diagonal covariance GMM fit to test set (12 mixture components).*

words, in the directions of the data dimensions). This means that the correlations between dimensions that were modelled by full covariance Gaussians and resulted in ellipses that could be shaped in any direction (seen in figures 2.5 and 2.6) have to be modelled by additional diagonal covariance mixture components.

This implies that more mixture components will give a better result. After some experimenting a good approximation is found with sixteen components. This result is shown in figures 2.9 and 2.10. Now we need to score against sixteen diagonal covariance Gaussians

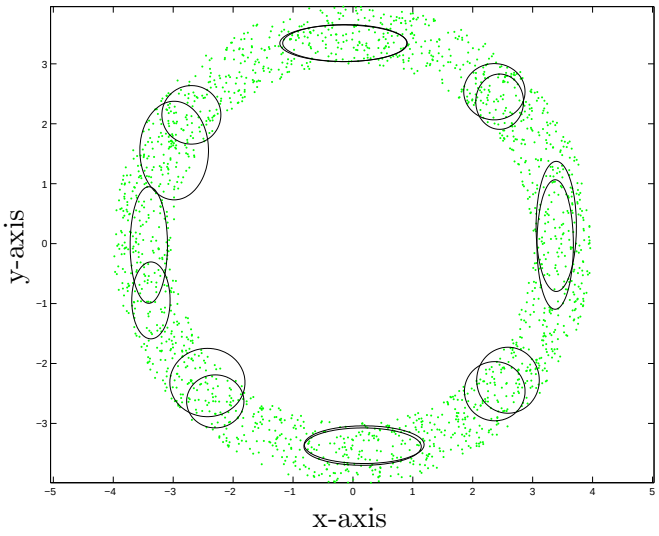


Figure 2.9: *Top view of a diagonal covariance GMM fit to test set (16 mixture components).*

for each feature. It is faster to calculate the likelihood for a diagonal covariance Gaussian than for a full covariance Gaussians. We do not have to invert any matrices and for each mixture component all the information that are not on the diagonal are not needed for any calculations. Considering that in this three dimensional case we removed more than half of the stored parameters for each matrix and only added a few mixture components, it results in an overall increase in calculation speed. This shows why diagonal covariance Gaussians are the dominant PDF used in GMMs in ASR systems today [23].

Because there is no matrix inversion, this method can model systems with less data better than the full covariance Gaussians can. More mixture components over less data means a better realization of the data and better modelling of individual clusters in the data. It also means that one can get a better approximation with diagonal covariance Gaussians than full covariance Gaussians when data is scarce.

One can also scale the diagonal covariance GMM easily, because adding or removing one mixture component does not equate in a large change in the model complexity. Implementing this method is also very easily done as shown in section 2.3.2. It requires only a few equations to change from the full covariance Gaussian model. When data is scarce and/or speed is an

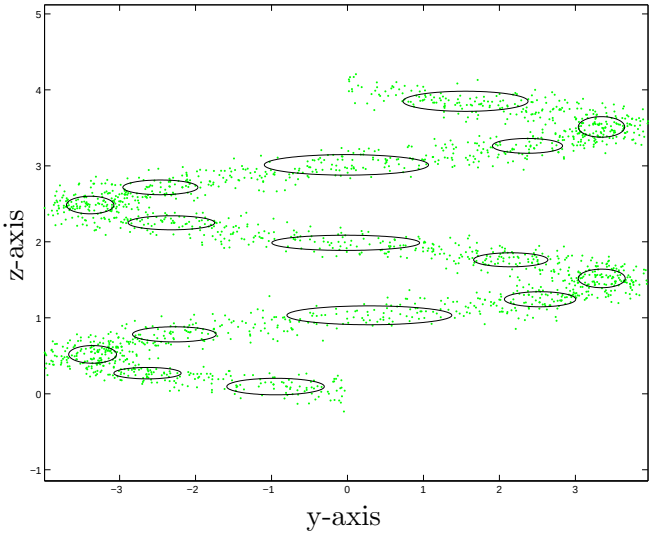


Figure 2.10: *Side view of a diagonal covariance GMM fit to test set (16 mixture components).*

issue, this method is preferable.

2.4 The Spherical Covariance Gaussian PDF

2.4.1 Theory

The spherical covariance Gaussian PDF is a further simplification of the diagonal covariance Gaussian PDF. It makes the same assumption that there is no correlation between dimensions and adds the assumption that all dimensions are equally distributed. The covariance matrix in equation 2.2 becomes:

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma^2 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \sigma^2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \sigma^2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \sigma^2 & 0 \\ 0 & 0 & 0 & \cdots & 0 & \sigma^2 \end{bmatrix}$$

The values on the diagonal are equal and is equal to the mean of all the values of the diagonal of the original covariance matrix it is derived from. It is intuitive that only the value of σ^2 needs to be stored.

This again only influences how the covariance is estimated and the mean stays the same

as in equation 2.3. For estimation of the covariance we replace equation 2.4 with:

$$\hat{\sigma}^2 = \frac{1}{D(N-1)} \sum_{n=1}^{N} \|\mathbf{x}_n - \hat{\boldsymbol{\mu}}\|^2 \tag{2.14}$$

where $\|\mathbf{x}\|^2 = \mathbf{x}^T\mathbf{x}$.

The log likelihood now becomes:

$$\log_e(p(\mathbf{x}|\boldsymbol{\mu}, \sigma^2)) = -\frac{\|\mathbf{x} - \boldsymbol{\mu}\|^2}{2\sigma^2} - \frac{D}{2}\log_e(2\pi\sigma^2) \tag{2.15}$$

2.4.2 Implementation

To implement the spherical covariance Gaussian PDF, all one needs to do is implement equations 2.3, 2.14 and 2.15. Like the the previous methods this is a closed loop solution and only needs to be computed once for a static system.

The answer to the likelihood in equation 2.15 has the same interpretation as the previous two methods. This method is simpler even than the diagonal covariance Gaussian PDF. For the likelihood of the spherical covariance Gaussian the same rules apply than for the previous two PDFs: the likelihoods of a set of features are summed to get the answer for the whole set and models are scored and compared the same way.

The implementation of the spherical covariance GMM is similar to the diagonal covariance GMM. The EM algorithm is still used and the E-step remains the same, only the definition of $p(\mathbf{x}_n|k)$ changes accordingly.

In the M-step, only the calculation of the covariance is changed to the following:

$$\hat{\sigma}_k^2 = \frac{\sum_{n=1}^{N} P(k|\mathbf{x}_n)\|\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k\|^2}{D\sum_{n=1}^{N} P(k|\mathbf{x}_n)} \tag{2.16}$$

The EM loop is repeated until the total score for the GMM converge.

2.4.3 Strengths and Weaknesses

Again we train a GMM to fit the dataset in figure 2.1. First we train for sixteen mixture components, the same number we needed to get a good representation with the diagonal covariance GMM. The results are shown in figures 2.11 and 2.12.

We can see that sixteen mixture components are not enough for a good representation. The spherical covariance Gaussians are restricted to stay circular (as all dimensions have the same covariance). This means more components are needed to fit the data in the test set. When doubling the number of components we get the results in figures 2.13 and 2.14.

These results show that spherical covariance GMMs need a large number of mixture components to represent complex data patterns. Although they are very fast to calculate, they model such a little amount of information that the number needed to represent data is very high. As mentioned before, this is because all the dimensions of the spheres are the

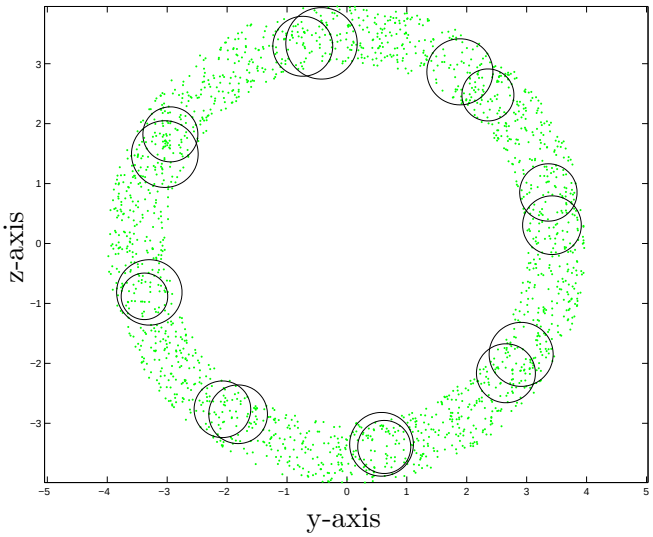


Figure 2.11: *Top view of a spherical covariance GMM fit to test set (16 mixture components).*

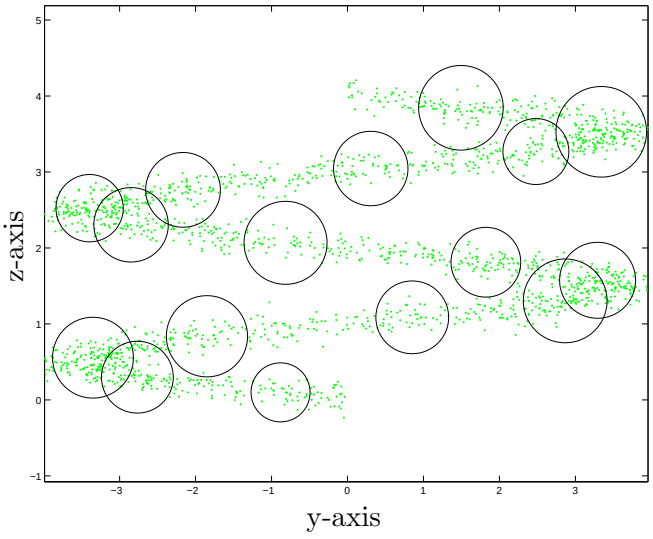


Figure 2.12: *Side view of a spherical covariance GMM fit to test set (16 mixture components).*

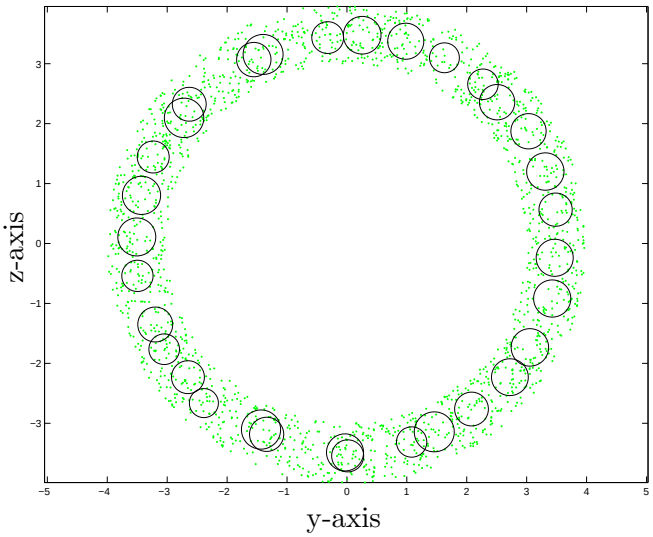


Figure 2.13: *Top view of a spherical covariance GMM fit to test set (32 mixture components).*

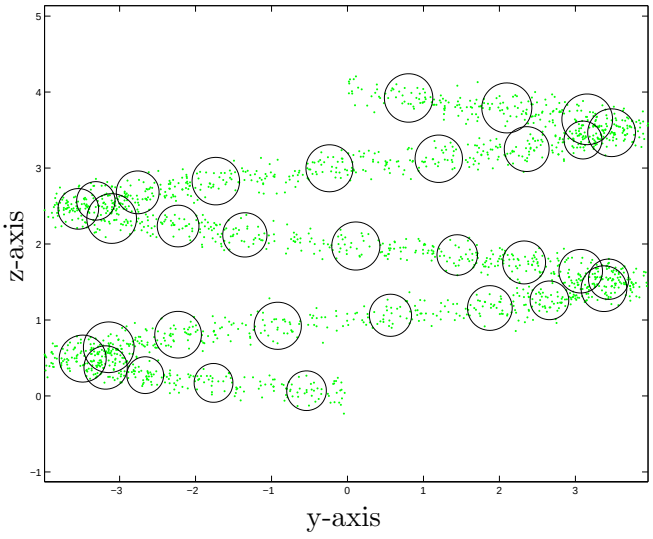


Figure 2.14: *Side view of a spherical covariance GMM fit to test set (32 mixture components).*

same. Not only do we need to model the differences between dimensions, we also need to model the correlations between dimensions with extra mixture components.

The spherical covariance GMM is just as simple to implement as the previous two methods. It is, as previously mentioned, very fast to calculate and does not need a lot of data to calculate. Spherical covariance GMMs is thus useful where data is very scarce.

When removing or adding mixture components, as illustrated in the above figures, there is a small difference in model complexity. In this case the difference is so small that many components needs to be removed or added to have a recognisable difference.

2.5 Summary

In this chapter we discussed three distribution models commonly used in GMMs today:

- The full covariance Gaussian PDF
- The diagonal covariance Gaussian PDF
- The spherical covariance Gaussian PDF

For each method we discussed the theory, implementation issues and strengths and weaknesses.

For the full covariance Gaussian PDF we found that:

* It is based on the classic multivariate Gaussian implementation
* It is easy to implement the equations into a non-iterative algorithm
* It is mathematically very expensive to compute as it needs multiplications with every member of the full covariance matrix.
* It needs only a few mixture components to model complex data
* It is not very scalable by adjusting the number of mixture components, as each component needs a large quantity of training data
* It is easy to over train when using a large number of mixture components
* It needs a lot of training data for each mixture component to give significant results

For the diagonal covariance Gaussian PDF we found that:

* It is a simplification of full covariance Gaussian implementation
* It is easy to implement the equations into a non-iterative algorithm
* It is mathematically less expensive to compute than the full covariance Gaussian PDF and much faster

* It needs more mixture components than the full covariance GMM to model complex data
* It is more scalable via adjusting the number of mixture components than the full covariance GMM as each component does not need as much training data
* It does not over train as easily and can handle a larger number of mixture components
* It does not need as much training data for each mixture component to give significant results

For the spherical covariance Gaussian PDF we found that:

* It is a simplification of the diagonal covariance Gaussian implementation
* It is easy to implement the equations into a non-iterative algorithm
* It is mathematically less expensive to compute than the diagonal covariance Gaussian PDF and faster
* It needs a large number of mixture components, even more than the diagonal covariance GMM, to model complex data
* It is scalable by adjusting the number of mixture components, but needs to add or remove a significant number of components to have a noticeable effect
* It is very hard to over train as it needs a large number of mixture components to get significant results
* It needs very little data per mixture component to build a significant model

From the above we can see the full covariance Gaussian is the more precise model of the three. It is very slow to compute however. On the other end of the scale, the spherical covariance Gaussian is the least precise model, but is very fast to compute.

A compromise can be found in the diagonal covariance Gaussian. It is relatively fast to compute and models a reasonable amount of statistical information. It is easy to see why it is the most used PDF in ASR today, as it gives a good balance between speed and accuracy.

Chapter 3

A Newer Generation of More Flexible Gaussian Models

3.1 Introduction

In this chapter we discuss three newly implemented models. These models have all been implemented to address some of the issues of the classic models discussed in chapter 2. The models discussed in this chapter are:

1. the sparse covariance Gaussian distribution model
2. the Probabilistic Principal Component Analysis (PPCA) covariance Gaussian distribution model
3. the Factor Analysis (FA) covariance Gaussian distribution model.

We first discuss some background needed for understanding the PPCA- and FA models. We discuss a general linear transform for dimension reduction followed by a discussion on Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). This is followed by a discussion on each of the above mentioned distribution models. As previously we discuss the relevant theory, implementation and the strengths and weaknesses of each model.

To illustrate the practical functioning of each model, we again use the same test set as in the previous chapter. This test set can be seen in figures 2.1 and 2.2 on page 16.

Again we train a GMM using each distribution model for the mixture components. We compare how these methods fare against those in chapter 2.

3.2 A General Linear Transform for Dimension Reduction

Most of the time we work with data of a high dimension. After feature extraction a dimensionality reduction transform is usually applied to minimise the dimensions and still retain

most of the information [25]. In general, when we want to reduce the dimensions of a dataset we use the following linear model:

$$\mathbf{v} = \mathbf{W}\mathbf{h} + \boldsymbol{\mu} + \boldsymbol{\epsilon} \tag{3.1}$$

where $\mathbf{v}$ is the original high dimensional representation, $\mathbf{W}$ is the linear transform matrix, $\mathbf{h}$ is the hidden lower dimension distribution used to generate $\mathbf{v}$ (known as the latent or hidden variables), $\boldsymbol{\mu}$ is a vector that determines the origin of the coordinate system and $\boldsymbol{\epsilon}$ is a Gaussian random variable with mean of $\mathbf{0}$ and a covariance matrix of $\boldsymbol{\Psi}$ [7]. The assumption is made that the variances of the discarded dimensions are negligible, or contained in $\boldsymbol{\epsilon}$.

3.2.1 PCA

We look as an example at PCA. Also known as the Karhunen-Loéve Transform (KLT), the PCA transform has been used extensively in pattern recognition. For PCA, the values of equation 3.1 are defined as:

$$\mathbf{W} = \mathbf{U}_M \tag{3.2}$$

with $\mathbf{U}_M$ being the eigenvectors corresponding to the M biggest eigenvalues of the covariance matrix of $\mathbf{v}$. The number of eigenvectors to keep corresponds to the number of dimensions chosen for $\mathbf{h}$. The value of $\boldsymbol{\epsilon}$ is such that $\boldsymbol{\Psi}$ is a spherical covariance matrix (see section 2.4). The value on the diagonals of $\boldsymbol{\Psi}$ is:

$$\sigma^2 = \frac{1}{D-M}\sum_{i=M+1}^{D}\lambda_i \tag{3.3}$$

with D the total number of dimensions. This value is basically the mean of the remaining eigenvalues. Because the remaining eigenvalues are the smallest, the $\boldsymbol{\epsilon}$ term is discarded. Now we use $\mathbf{W}$ to get a lower dimensional representation of $\mathbf{v}$:

$$\hat{\mathbf{t}} = \mathbf{W}^T\mathbf{v} \tag{3.4}$$

It is obvious that some information is lost in equation 3.4 because of the removal of $\boldsymbol{\epsilon}$. This is acceptable if the remaining eigenvalues are small in relation to those kept. The error made is quantified by the sum of the remaining eigenvalues. Although $\boldsymbol{\mu}$ is also disregarded, this only has the effect of the mean being transformed to a new value. Before calculating the PDF models, this change in the mean value does not make a difference.

PCA is used extensively in PR today and is a basic part of LDA, which is used for dimension reduction in speech signals [25].

$\mathbf{W}$ is also commonly chosen as:

$$\mathbf{W} = \mathbf{U}_M\boldsymbol{\lambda}_M^{1/2} \tag{3.5}$$

with $\boldsymbol{\lambda}_M$ being the M largest eigenvalues. When using this $\mathbf{W}$, $\mathbf{h}$ is assumed to have a covariance matrix equal to the identity matrix. This is because $\mathbf{W}$ is scaled to normalise the variance.

3.2.2 LDA

LDA is commonly used for dimensionality reduction in speech systems. The LDA transform has the advantage over the PCA transform of also considering cluster information. When the clusters in the data are known (either by choice or an automatic clustering technique), the LDA transforms the data to ensure maximum separability between these clusters (see figure 1.1 on page 3).

First we need to calculate the average covariance matrix of all the clusters. Each cluster is given a prior probability by calculating its significance to the total dataset. Each cluster covariance matrix is weighted by this prior probability and added up to get the average covariance matrix:

$$\mathbf{S}_W = \sum_{i=1}^{C} P_i \boldsymbol{\Sigma}_i \tag{3.6}$$

$$\boldsymbol{\mu} = \sum_{i=1}^{C} P_i \boldsymbol{\mu}_i \tag{3.7}$$

where $\mathbf{S}_W$ is the average covariance matrix, C is the number of clusters, P_i is the prior probability of the ith cluster, $\boldsymbol{\Sigma}_i$ is the covariance matrix of the ith cluster and $\boldsymbol{\mu}_i$ is the mean of the ith cluster. $\boldsymbol{\mu}$ should be equal to the mean of the whole dataset.

Next we calculate the covariance matrix of the dataset containing only the cluster mean values. This is calculated around the data mean $\boldsymbol{\mu}$ and weighted by the cluster prior probabilities:

$$\mathbf{S}_B = \sum_{i=1}^{C} P_i (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T \tag{3.8}$$

We calculate the whitening transform (so called, because white noise has unity variance and the purpose of this transform is to make the total covariance equal the unity matrix):

$$\mathbf{B} = \mathbf{U}_W \boldsymbol{\lambda}_W^{-1/2} \tag{3.9}$$

where $\mathbf{U}_W$ is the eigenvectors of $\mathbf{S}_W$ and $\boldsymbol{\lambda}_W$ is the diagonal matrix containing the eigenvalues of $\mathbf{S}_W$.

Using $\mathbf{B}$ as a transformation matrix to transform the data, we effectively factor out the $\mathbf{S}_W$ covariance. This makes the average covariance of the clusters equal to the unity matrix. If the assumption is made that all the clusters are similarly distributed, this transformation makes them all have a covariance matrix that is close to the identity matrix. This assumes each cluster contains no individual extra information in terms of variance.

This also means the covariance matrix of the mean values of the cluster is transformed by $\mathbf{B}$:

$$\mathbf{S}'_B = \mathbf{B}^T \mathbf{S}_B \mathbf{B} \tag{3.10}$$

Now we calculate the PCA transform of the $\mathbf{S}'_B$ covariance matrix to find the directions where the centroids of the clusters have the highest variance:

$$\mathbf{W}_{\text{PCA}} = \mathbf{U}_M \tag{3.11}$$

where $\mathbf{U}_M$ is the eigenvectors of $\mathbf{S}'_B$ corresponding to the M largest eigenvalues.

The LDA transform is thus defined as:

$$\mathbf{W}_{\text{LDA}} = \mathbf{B}\mathbf{W}_{\text{PCA}} \tag{3.12}$$

Transforming the dataset with $\mathbf{W}_{\text{LDA}}$ will transform the average cluster covariance matrix to the identity matrix and discard the dimensions that have the least information regarding the cluster positioning.

3.3 The Sparse covariance Gaussian PDF

3.3.1 Theory

The sparse covariance Gaussian PDF fits in the gap between the full covariance Gaussian and the diagonal covariance Gaussian representations. The sparse covariance Gaussian, like the diagonal, ignores some of the off-diagonal values of the covariance matrix. Unlike the diagonal, it does not discard all these values. Instead it uses a method of defining 'blocks' of dimensions that are correlated to each other. To save computation time, the size of these blocks are limited to a predetermined value. Another limit is the minimum value of correlation coefficients to accept [8, 10].

With these two limits set in place, the entries in the covariance matrix of the most correlated dimensions are kept, while the rest are assumed to be zero. These blocks are also used to simplify the scoring by only using the relevant information for calculation.

The mean vector is still in the same form as the previous methods. A typical $\mathbf{\Sigma}$ matrix might look like the following:

$$\mathbf{\Sigma} = \begin{bmatrix} c_{00} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & c_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & c_{22} & 0 & c_{24} & 0 & 0 & c_{27} & c_{28} \\ 0 & 0 & 0 & c_{33} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & c_{42} & 0 & c_{44} & 0 & 0 & c_{47} & c_{48} \\ 0 & 0 & 0 & 0 & 0 & c_{55} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & c_{66} & 0 & 0 \\ 0 & 0 & c_{72} & 0 & c_{74} & 0 & 0 & c_{77} & c_{78} \\ 0 & 0 & c_{82} & 0 & c_{84} & 0 & 0 & c_{87} & c_{88} \end{bmatrix}$$

We can see that all the values for the diagonal and only some of the off diagonals are kept. The off diagonals that are kept are stored as blocks that are correlated. The only block for

the above matrix is $\{2, 4, 7, 8\}$. It means that all these dimensions are correlated with each other as if they are part of a full covariance matrix.

The dimensions between blocks are modelled like they are part of a diagonal covariance matrix. Thus if we arrange the dimensions in such a way that the correlated dimensions were next to each other, the covariance matrix would only have non-zero values around the main diagonal:

$$\boldsymbol{\Sigma} = \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} & 0 & 0 & 0 & 0 & 0 \\ c_{10} & c_{11} & c_{12} & c_{13} & 0 & 0 & 0 & 0 & 0 \\ c_{20} & c_{21} & c_{22} & c_{23} & 0 & 0 & 0 & 0 & 0 \\ c_{30} & c_{31} & c_{32} & c_{33} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & c_{44} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & c_{55} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & c_{66} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_{77} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_{88} \end{bmatrix}$$

Here the above matrix is the same as the previous one, but now the dimensions have been arranged in such a way that dimensions correlated are $\{0, 1, 2, 3\}$. We only need to store the diagonal values and the indices and covariance matrix entries for each block, instead of the whole matrix.

When trying to estimate this model from training data we still use equations 2.3 and 2.4, repeated here:

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \tag{3.13}$$

$$\hat{\boldsymbol{\Sigma}}_{\text{full}} = \frac{1}{N-1}\sum_{n=1}^{N}(\mathbf{x}_n - \hat{\boldsymbol{\mu}})(\mathbf{x}_n - \hat{\boldsymbol{\mu}})^T \tag{3.14}$$

This would give us a full Gaussian representation, which is the starting point for calculating the sparse covariance Gaussian. After the above steps, we take $\hat{\boldsymbol{\Sigma}}_{\text{full}}$ and calculate the correlation coefficient matrix with the following equation:

$$\rho_{xy} = \frac{c_{xy}}{\sigma_x \sigma_y} \tag{3.15}$$

with c_{xy} being the covariance between the x and y dimensions, σ_x being the standard deviation of the x^{th} dimension and $-1 \geq \rho_{xy} \geq 1$ quantifying the correlation between x and y [22]. Next we set all the diagonals of the correlation coefficient matrix to zero (as they will be equal to one) and set all correlation coefficients below a chosen threshold to zero.

Next we take each correlation coefficient, from the highest to the lowest, and construct the blocks accordingly. The blocks are seen as individual sets of dimensions that are correlated to each other as if they were part of a full covariance matrix. When two correlated dimensions are not yet in a block, a new block is created. If only one of the correlated dimensions is

already in a block, the other dimension is added to that block. When the two correlated dimensions are in two separate blocks, these blocks are merged. Before these operations are completed, the size of the completed block is checked. If it exceeds a maximum value determined initially, the operation is not completed and the next correlation coefficient is evaluated.

The resultant blocks define what values should be saved from the full covariance matrix $\hat{\mathbf{\Sigma}}_{\text{full}}$. The diagonal is stored in a vector and the block indices and their values are stored in two sets of vectors.

To calculate the log likelihood we can optimise the equation

$$\log_e(p(\mathbf{x}|\boldsymbol{\mu}, \mathbf{\Sigma})) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}) - \frac{D}{2}\log_e(2\pi) - \frac{1}{2}\log_e(|\mathbf{\Sigma}|) \tag{3.16}$$

in such a way that only the non-zero values of $\hat{\mathbf{\Sigma}}$ are considered. We can do this by using the information stored in the blocks. We know the indices of the nonzero values and their values. We can invert the sparse matrix (as a full matrix) and store the inverse in sparse format with the same blocks as calculated previously. The determinant of the sparse matrix can be calculated simultaneously and stored.

When calculating $(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$ in equation 3.16 we only need to multiply and add the non-zero values of the inverse with the corresponding values in $(\mathbf{x}-\boldsymbol{\mu})$. The index for the vector $(\mathbf{x}-\boldsymbol{\mu})$ to use on either side of the non-zero covariance value is determined by the row and column indices of the value stored in the block (due to the way matrix multiplication works [19]). Once we have done this for all the non-zero values (including the diagonal values) the sum of all these values can be substituted for $(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})$ in equation 3.16.

It is obvious that this method is more expensive than the diagonal Gaussian since we still need to calculate more parameters of the covariance matrix than just the diagonal. The multiplications between the test set and the inverse matrix has been reduced, so it is less expensive than a full Gaussian. One can further simplify the calculation by taking into account that the covariance matrix is symmetrical.

3.3.2 Implementation

The first part of implementation of the sparse covariance Gaussian PDF is exactly like implementing the full Gaussian PDF. We use equations 3.13 and 3.14. Then using the algorithm described in the previous section, the parameters for the sparse covariance is calculated. This is also a closed loop solution.

If after calculating the statistics, one can store information such as the determinant of the covariance (a single scalar) and the inverse matrix in sparse form as this information is used for scoring.

The likelihood described above has the same interpretation as with the previous PDFs. Likelihoods for each vector in a set can be summed to get the total likelihood for the set and the answers are weighed against each other in the same way.

Using the sparse covariance Gaussian PDF in a GMM is much the same as before. The statistics are calculated as for a full matrix.

The E-step:

$$P(k|\mathbf{x}_n) = \frac{p(\mathbf{x}_n|k)P(k)}{p(\mathbf{x}_n)} \tag{3.17}$$

The M-step:

$$\hat{P}(k) = \frac{1}{N}\sum_{n=1}^{N} P(k|\mathbf{x}_n) \tag{3.18}$$

$$\hat{\boldsymbol{\mu}}_k = \frac{\sum_{n=1}^{N} P(k|\mathbf{x}_n)\mathbf{x}_n}{\sum_{n=1}^{N} P(k|\mathbf{x}_n)} \tag{3.19}$$

$$(\hat{\boldsymbol{\Sigma}}_{\text{full}})_k = \frac{\sum_{n=1}^{N} P(k|\mathbf{x}_n)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}_n - \hat{\boldsymbol{\mu}}_k)^T}{\sum_{n=1}^{N} P(k|\mathbf{x}_n)} \tag{3.20}$$

We then get the sparse form of each $(\hat{\boldsymbol{\Sigma}}_{\text{full}})_k$ as described in the previous section.

The E-step and M-step are iteratively calculated until the likelihood of the GMM stabilises.

3.3.3 Strengths and Weaknesses

We train the sparse covariance GMM to fit our test set again. It is intuitive that the sparse covariance Gaussian PDF fits in between full and diagonal covariance PDFs. First we compare it with the full covariance Gaussian. The full covariance Gaussian had a good representation with twelve mixture components. We train a sparse covariance GMM with the constraints of two dimensions per block (if we take three dimensions in this case, we would represent a full covariance Gaussian, as the dataset only has three dimensions) and a minimum correlation coefficient value of 0.2. The results can be seen in figures 3.1 and 3.2.

We can see here that although it is not the best representation, it does a lot better than the standard diagonal covariance GMM. This is because many of the mixture components are correlated in one pair of dimensions (because we have a block size of two). This can be seen in figure 3.1 as there are quite a few ellipses that are not constrained only in the direction of the axes. One can also see in both figures ellipses that do indeed only shape in the direction of the axis. These are between dimensions that are not in the same (or in any) block.

In this case, each mixture component can model the dimensions with the highest correlation coefficients accurately (if above the threshold) and model the remaining dimensions as uncorrelated. Let us look at sixteen components, the number that gave an acceptable diagonal model. This is shown in figures 3.3 and 3.4.

We see that sixteen mixture components give a very nice representation. In fact, sixteen components are the lowest number of components that give a good representation. The sparse representation is much better than the diagonal one in figures 2.9 and 2.10. Especially

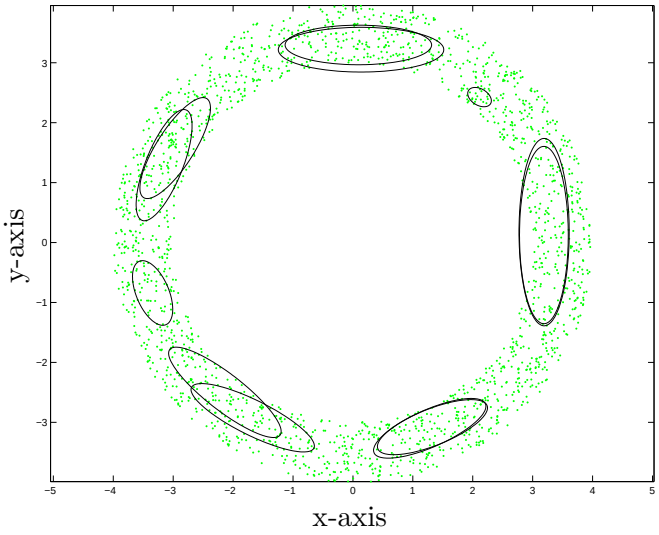


Figure 3.1: *Top view of a sparse covariance GMM fit to test set (12 mixture components).*

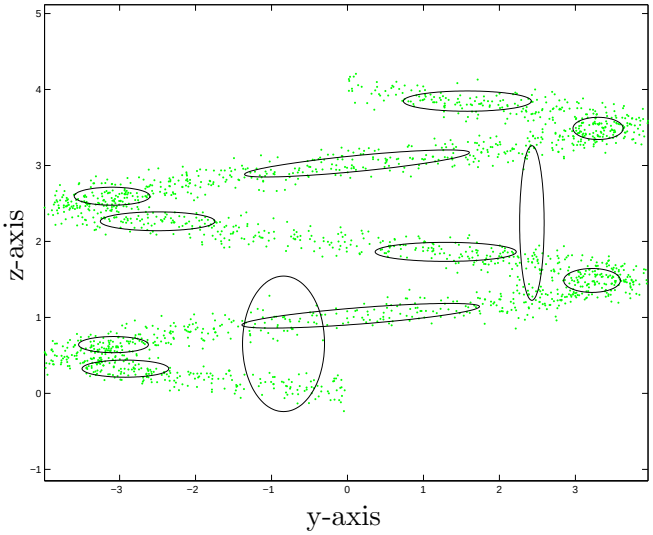


Figure 3.2: *Side view of a sparse covariance GMM fit to test set (12 mixture components).*

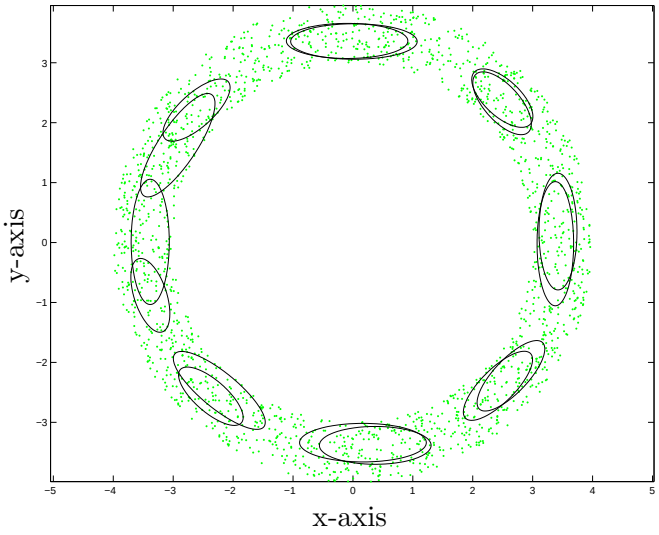


Figure 3.3: *Top view of a sparse covariance GMM fit to test set (16 mixture components).*

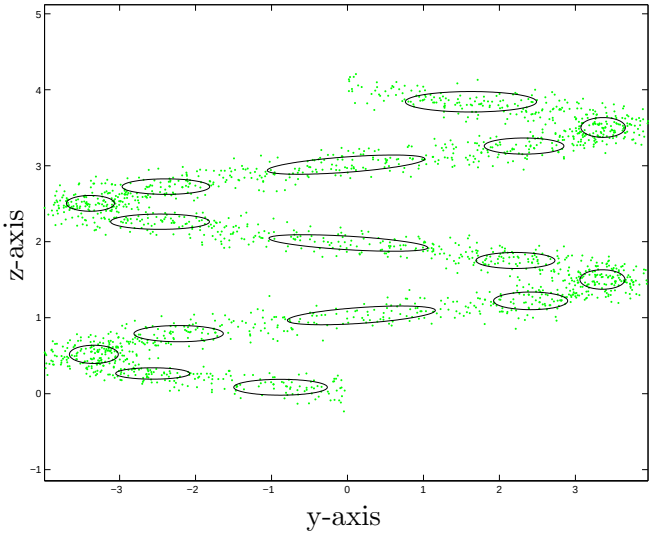


Figure 3.4: *Side view of a sparse covariance GMM fit to test set (16 mixture components).*

in the side view we see the sparse covariance Gaussian makes it possible to accurately model the steady incline of the 'spring'. From the top view one can also see the representation is much better because some mixture components model the correlation between dimensions.

Thus while the sparse covariance Gaussian is more complex than diagonal covariance Gaussian, it gives a much more accurate representation. As it should be faster than the full covariance Gaussian, it would fit in between the two. If one has some processing power to spare, but the number of mixture components one can use on the data is limited and a full covariance Gaussian is definitely too slow, the sparse covariance Gaussian is a perfect candidate. What makes it more useful is the fact that each component can be scaled in terms of the maximum block size and the minimum correlation coefficient. For large dimensional cases one can scale the PDF through a wide range between diagonal and full covariance. The speed increases over full covariance Gaussians are also much more apparent with higher dimensional cases, depending on how much information is kept.

Although more training data is needed per mixture component (due to the fact that more statistical data is recorded) than with the diagonal covariance Gaussian to build a good model, less is needed than for the full covariance case. Because all the unwanted elements have been removed by the time that inversion takes place, the data needed for a good model is proportional to the amount of data kept in the matrix. This method can therefor be scaled if the training data is less or more than desired.

The scalability of a sparse covariance GMM via the number of mixture components is also dependant on how much information is kept per component. Depending on one's needs, the ratio between mixture component complexity and number of components can be varied to suit the system. In this way, over training, due to using too many mixture components, can also be reduced.

Implementation of this method is quite complex, as discussed in section 3.3.1, however, when one needs a model that holds more information than the diagonal covariance Gaussian and the full covariance Gaussian is too expensive, this method is a good choice.

3.4 The PPCA Covariance Gaussian PDF

3.4.1 Theory

The Probabilistic Principal Component Analysis (PPCA) covariance Gaussian is another method that can hold a variable amount of information. It is based on the PCA method (discussed in section 3.2.1). The difference between PCA and PPCA is that the noise element $\boldsymbol{\epsilon}$ in equation 3.1 is not ignored. Thus instead of removing the information from the discarded dimensions, the information is generalised in $\boldsymbol{\epsilon}$. This ensures the result of the transform is still a valid PDF representation of the data [28].

When we choose the number of dimensions to keep as M, we can calculate the value of

σ^2 with equation 3.3. Now we can rewrite equation 3.1 as:

$$\mathbf{v} - \boldsymbol{\epsilon} = \mathbf{W}\mathbf{h} + \boldsymbol{\mu} \tag{3.21}$$

Now the value of $\mathbf{W}$ becomes:

$$\mathbf{W} = \mathbf{U}_M(\boldsymbol{\lambda}_M - \sigma^2\mathbf{I})^{1/2}\mathbf{R} \tag{3.22}$$

where $\mathbf{U}_M$ is the matrix of eigenvectors corresponding to the largest M eigenvalues of the covariance of $\mathbf{v}$ and $\boldsymbol{\lambda}_M$ is an diagonal matrix with the M largest eigenvalues on the diagonals. $\mathbf{R}$ is an arbitrary rotation matrix, i.e. it can be any orthogonal matrix. We can see that in this case, we have the variance information included in $\mathbf{W}$. We can now store the values of $\mathbf{W}$ and σ^2 instead of the whole covariance matrix. We can use these values to get an estimate of the covariance matrix:

$$\hat{\boldsymbol{\Sigma}} = \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I} \tag{3.23}$$

In the above equation $\mathbf{W}\mathbf{W}^T$ contains the covariance information of the retained dimensions. With $\sigma^2\mathbf{I}$ we add the noise term to get the final matrix representation of the estimated covariance matrix. We can see that only the retained dimension information is kept explicitly. The remaining dimensions are all generalised to the covariance matrix of $\boldsymbol{\epsilon}$.

When we estimate the statistics from the training data, we again use equations 3.13 and 3.14. We then use eigenvalue decomposition on $\boldsymbol{\Sigma}$ and equations 3.3 and 3.22 to get σ^2 and $\mathbf{W}$. We also define a parameter to limit the minimum eigenvalue for a principal component. It is a number between 0 and 1. For each principal component kept, the corresponding eigenvalue is evaluated as a ratio to the sum of all the eigenvalues. If this value is smaller than the minimum eigenvalue parameter, the principal component is disregarded.

To get the log likelihood we again use equation 3.16. We can take advantage of the lower dimensional space that is used to calculate $\boldsymbol{\Sigma}$ to find its inverse. If we define the $i \times i$ matrix $\boldsymbol{\Psi}_i = \sigma^2\mathbf{I}$ and rewrite equation 3.23, we get:

$$\hat{\boldsymbol{\Sigma}} = \boldsymbol{\Psi}_D(\mathbf{I} + \mathbf{W}\boldsymbol{\Psi}_M^{-1}\mathbf{W}^T) \tag{3.24}$$

where M is the number of the retained dimensions and D the number of the total dimensions. Using the *Matrix Inversion Lemma* [15], we get the inverse of $\hat{\boldsymbol{\Sigma}}$ as:

$$\hat{\boldsymbol{\Sigma}}^{-1} = (\mathbf{I} - \mathbf{W}(\boldsymbol{\Psi}_M + \mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T)\boldsymbol{\Psi}_D^{-1} \tag{3.25}$$

which, if we reintroduce σ^2, can be written as:

$$\hat{\boldsymbol{\Sigma}}^{-1} = \frac{\mathbf{I} - \mathbf{W}(\sigma^2\mathbf{I} + \mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T}{\sigma^2} \tag{3.26}$$

This inversion is faster to calculate, because $\mathbf{S} = (\sigma^2\mathbf{I} + \mathbf{W}^T\mathbf{W})$ is an $M \times M$ matrix.

We can further improve the speed of equation 3.16 by setting:

$$(\mathbf{x} - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) = \frac{(\mathbf{x} - \boldsymbol{\mu})^T(\mathbf{x} - \boldsymbol{\mu}) - (\mathbf{x}')^T\mathbf{S}^{-1}\mathbf{x}'}{\sigma^2} \tag{3.27}$$

where $\mathbf{x}' = \mathbf{W}^T(\mathbf{x} - \boldsymbol{\mu})$. The right-hand side is faster to calculate with a computer than the left-hand side as there are less element multiplications that take place. The larger the value of $(D - M)$, the more significant the speed increase. This makes a big difference, as these multiplications need to take place with every score calculation.

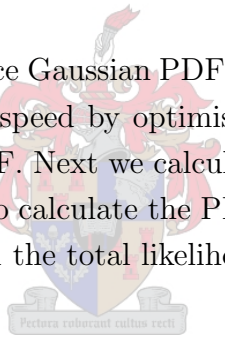
3.4.2 Implementation

As with the sparse covariance Gaussian PDF, the estimation of the PPCA covariance Gaussian PDF starts with equations 3.13 and 3.14. Then the PPCA covariance Gaussian PDF is estimated from the full covariance matrix as described above. The solution is closed loop and the calculations only have to be done once for a static training set.

To calculate the score against a PDF we need the inverse of the covariance matrix only. Matrix inversion is mathematically expensive, so to speed up scoring, we thus store S^{-1} instead of the covariance matrix. This reduces the time that would have been spent on matrix inversion.

The resultant likelihood has the same interpretation as the previous methods: the likelihoods of a set can be summed for the total set likelihood and comparisons are done in the same manner.

When using the PPCA covariance Gaussian PDF in a GMM, we first calculate the E-step as in equation 3.17. Here we gain speed by optimisations in calculating the likelihood for the PPCA covariance Gaussian PDF. Next we calculate the equations 3.18 to 3.20. The full covariance matrices are then used to calculate the PPCA equivalents as described in section 3.4.1. These steps are iterated until the total likelihood stabilises.



3.4.3 Strengths and Weaknesses

We again train a GMM containing PPCA covariance Gaussian PDFs to fit our test set. The modelling abilities of this method falls between the spherical covariance GMM (when all dimensions are discarded and generalised by σ^2) and the full covariance GMM (when only one dimension is discarded and thus fully described by σ^2). This method should be faster than the full covariance GMM (depending on how many dimensions are retained, because there is extra overhead in calculating the PPCA parameters) but slower than the diagonal covariance GMM.

For our test set, we choose to retain one dimension as important. If we chose none, we would have the same effect as a spherical covariance GMM. If we chose two it would be the same as full covariance Gaussian. One is thus the only logical option in the three dimensional case.

We choose the minimum eigenvalue parameter to be 0.667. The reason for this is, the more the mixture component resembles a spherical covariance Gaussian, the more the value of the eigenvalues will converge to the same value. With a spherical covariance Gaussian of three dimensions we have each eigenvalue at 0.333 of the sum of all the values. We choose

to only evaluate eigenvalues that are twice this size (to allow some variation).

First the twelve mixture component case, one which worked well for the full covariance GMM. The results can be seen in figures 3.5 and 3.6.

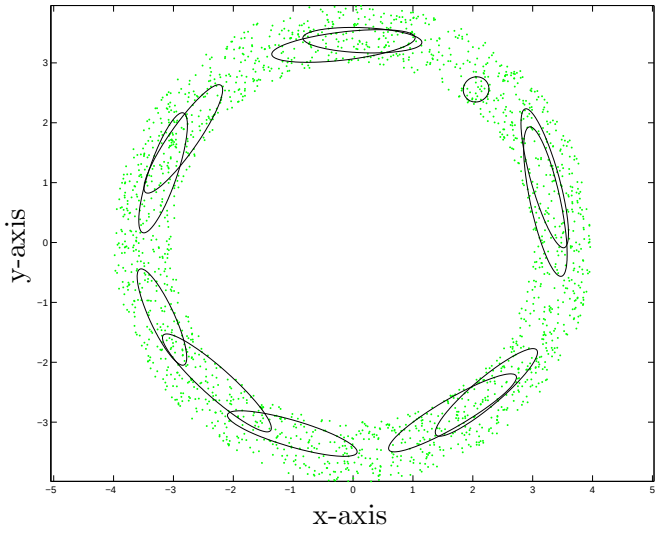


Figure 3.5: *Top view of a PPCA covariance GMM fit to test set (12 mixture components).*

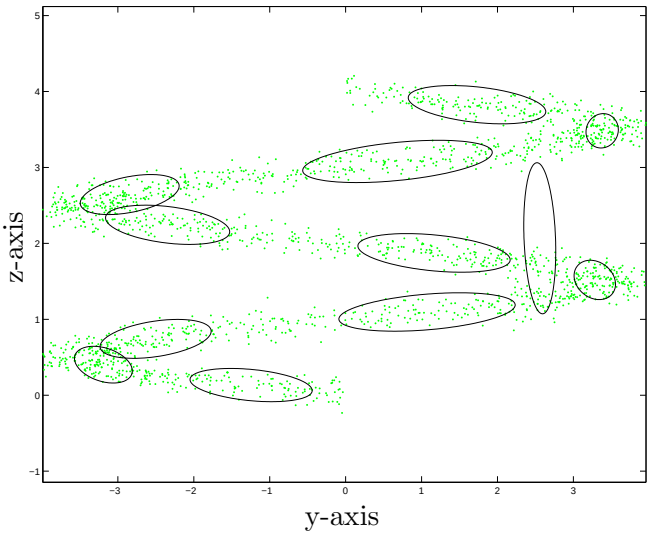


Figure 3.6: *Side view of a PPCA covariance GMM fit to test set (12 mixture components).*

Again we can see that this PDF gives a better model than the diagonal covariance PDF did, although it seems the sparse covariance GMM did better. The reason for this method being better than the diagonal covariance method is because the full covariance information

of the principal directions are used. These directions are also not limited to the directions of the axes. One can see in figures 3.5 and 3.6 that the mixture components are, like full covariance Gaussians shaped in different directions.

One may make the assumption that this method works the same way as the LDA. While this is true in this case, the effect of an LDA is applied to each mixture component individually, rather than over the whole dataset.

The weakness of this model is the fact that the directions perpendicular to the ones explicitly modelled are essentially spherically distributed. We can see this in figure 3.5 where some of the distributions seem too 'narrow'. This is because of the fact that variances in two directions are being generalised by one value. It means that the two remaining dimensions are 'locked' to each other, where the sparse covariance Gaussian is at worse diagonal in some dimensions.

Next we look at sixteen mixture components, a value that gave acceptable results for the diagonal covariance GMM and good results for the sparse covariance GMM. In figures 3.7

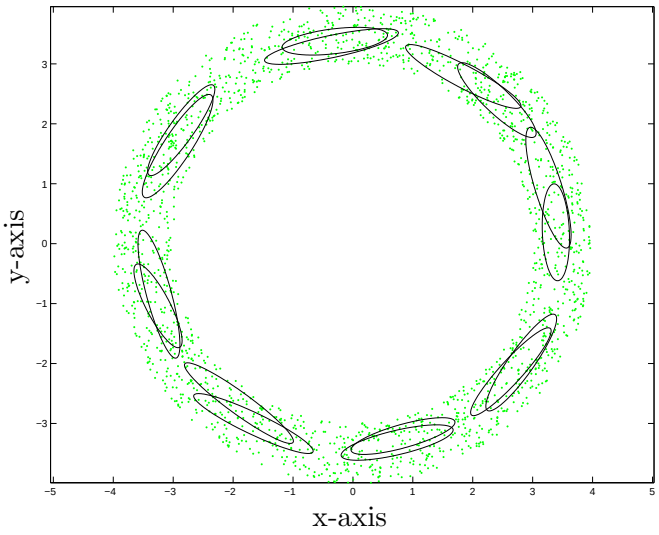


Figure 3.7: *Top view of a PPCA covariance GMM fit to test set (16 mixture components).*

and 3.8 we can see that this method gives a better representation of the test set than the diagonal covariance Gaussian. Although we do see an improvement from the twelve mixture component case, some components still do not fit the data exactly due to the spherical effect. However, all the components are aligned in the direction of the data, unlike the sparse covariance Gaussian where some components are still diagonal. This is because the sparse covariance Gaussian is limited in its modelling of correlations between dimensions and have to behave like a diagonal distribution in some dimensions. The PPCA covariance Gaussians can have its principal direction preserved and thus keep the correlation information in this direction over the initial axes.

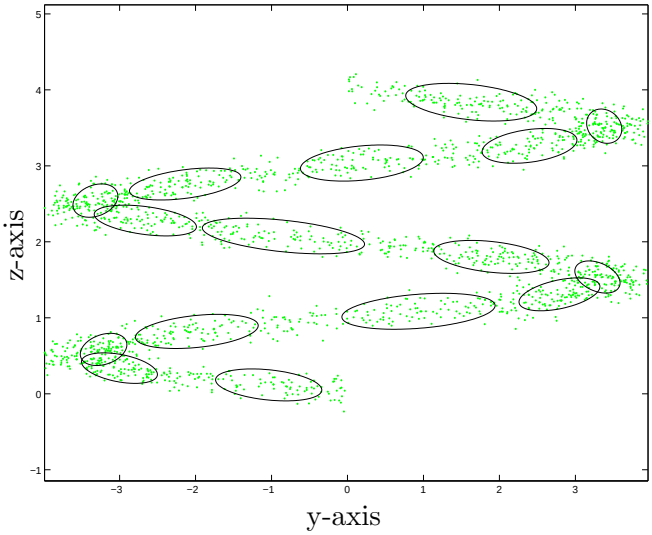


Figure 3.8: *Side view of a PPCA covariance GMM fit to test set (16 mixture components).*

With enough mixture components the spherical generalised dimensions can be covered by groups of components as is seen in figure 3.7. This shows that comparing the PPCA covariance method directly to the sparse covariance method is difficult, as they both try to keep important information based on their own assumptions. It is difficult to compare the amount of useful information modelled by each one. This can be seen in the above examples, where with too few mixture components, the sparse version modelled the data better. With more components they both seem to approximate good models of the data.

This example would suggest that the sparse covariance PDF works better with systems that have a limited number of mixture components than the PPCA covariance PDF. This conclusion cannot be taken as a fact without more testing though.

This model, like the sparse covariance Gaussian PDF, can be scaled for each mixture component depending on how many principal directions are kept in the transform. The larger the dimension of the dataset, the bigger the scaling range. The speed increase over the full covariance Gaussian is also much more apparent when the dataset has a higher dimension.

The PPCA mixture components also need less data than a full covariance Gaussian to get a good representation. The data needed per component for a PPCA covariance Gaussian depends on how many principal directions are explicitly modelled. This defines the size of S, the only matrix that needs to be inverted. Like the sparse covariance Gaussian, one can adjust the parameters for this model depending on the scarceness of data in the system.

The scalability of the GMM via the adjustment of the number of mixture components is also dependant on the number of principal components kept per component. The ratio between mixture component complexity and the number of components can be fine tuned

depending on the system. We can also avoid over training in this way.

Both these methods are an alternative to diagonal covariance PDFs when one wants to include more information in the model without having the speed penalty of the full covariance Gaussian PDF.

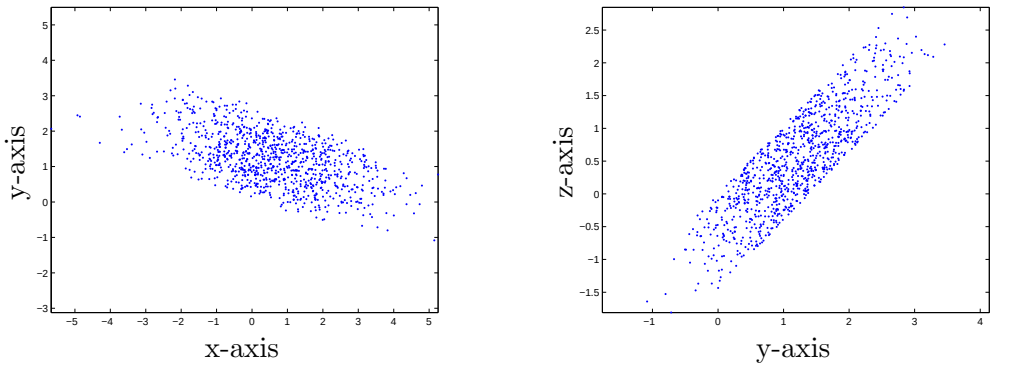
3.5 The FA Covariance Gaussian PDF

3.5.1 Theory

The Factor Analysis (FA) covariance Gaussian PDF is yet another type of PDF that can hold a variable amount of information. It is based on the general linear transform discussed in section 3.2. In the PPCA case, the assumption was made that $\boldsymbol{\epsilon}$ in equation 3.1 was spherically distributed. With the FA transform, we make the assumption that $\boldsymbol{\epsilon}$ is diagonally distributed.

Each direction will now have its own noise variance (the directions are defined in terms of the axis used for the original data). In the case of the PPCA, we were fortunate enough that the eigen-decomposition of the covariance matrix gave us a valid solution for $\mathbf{W}$ [7, 11]. This is no longer the case for the FA transform. The reason for this is because of the generalisation of the noise variance in the PPCA method. When we calculated $\mathbf{W}$, the discarded dimensions all were assumed to have the same variance. This means that the added noise element is the same in every direction (like a sphere), and act as a scale factor.

With the FA transform however, the added noise is diagonal and each value corresponds to one of the main axes in which the data is defined. This means that if the information retained in the $\mathbf{W}$ matrix is not perpendicular with the axes, the added noise would scale differently with each axis and distort the data, changing the principal directions. The principal directions of $\mathbf{W}$ would therefor not correspond to the eigenvectors of the covariance matrix.

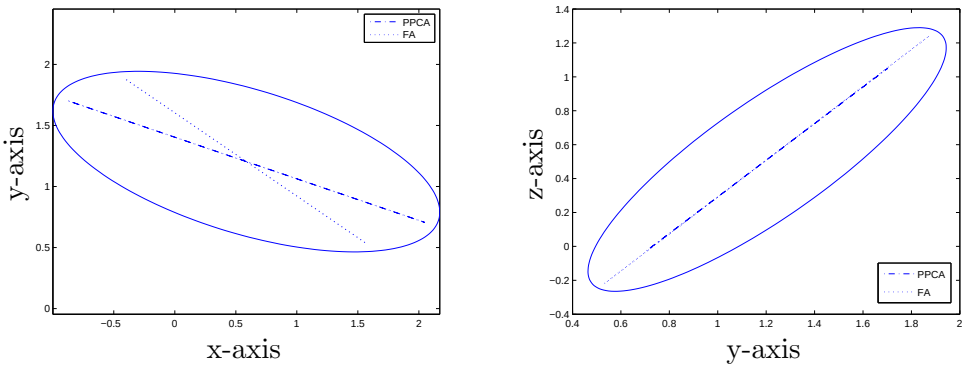


(a) X-Y axis view of an example dataset. (b) Y-Z axis view of an example dataset.

Figure 3.9: *Top and side views of an example dataset.*

As an example we look at the synthetic dataset pictured in figures 3.9a and 3.9b. We

represent the dataset with a full Gaussian and observe the retained information with one principal component (with the PPCA) and with one factor (with the FA) in figures 3.10a and 3.10b.



(a) Representation of figure 3.9a. (b) Representation of figure 3.9b.

Figure 3.10: *Full Gaussian representation of figure 3.9 with principal components and factors. Note that in 3.10a we can clearly see the PPCA and FA methods deviating.*

We see that in figure 3.10a the difference between the direction of the factor is different than that of the principal axes of the system. When the PPCA covariance Gaussian is calculated by adding the spherical noise element, the information kept by the principal direction is uniformly scaled in all the directions (as a sphere is uniform in all directions) giving the final Gaussian representation the same principal direction.

With the FA covariance Gaussian however, the noise element is diagonal. Because the added diagonal matrix is defined by the same axes as the dataset, each value on the diagonal matrix is a scaling factor corresponding to the axis it represents. This causes the principal direction of the final FA covariance Gaussian to be different from the direction represented by the retained factor (see figure 3.11.

Principal component analysis is no longer a solution to this problem. We look again at equation 3.21. We want to find a solution for this equation with $\boldsymbol{\epsilon}$ being distributed diagonally. Since we have no easy solution, Maximum Likelihood (ML) estimation is used to find the solution. For this we again use a version of the EM algorithm [1, 14].

First the E-step. Note that we assume that $\boldsymbol{\mu} = \mathbf{0}$. We calculate the expected value of $\mathbf{h}$ given $\mathbf{v}_i$:

$$\mathrm{E}(\mathbf{h}|\mathbf{v}_i) = \boldsymbol{\beta}\mathbf{v}_i \tag{3.28}$$

where $\boldsymbol{\beta} = \mathbf{W}^T(\boldsymbol{\Psi} + \mathbf{W}\mathbf{W}^T)^{-1}$. Furthermore we need to calculate the second moment of $\mathbf{h}$ given $\mathbf{v}_i$:

$$\mathrm{E}(\mathbf{h}\mathbf{h}^T|\mathbf{v}_i) = \mathbf{I} - \boldsymbol{\beta}W + \boldsymbol{\beta}\mathbf{v}_i\mathbf{v}_i^T\boldsymbol{\beta}^T \tag{3.29}$$

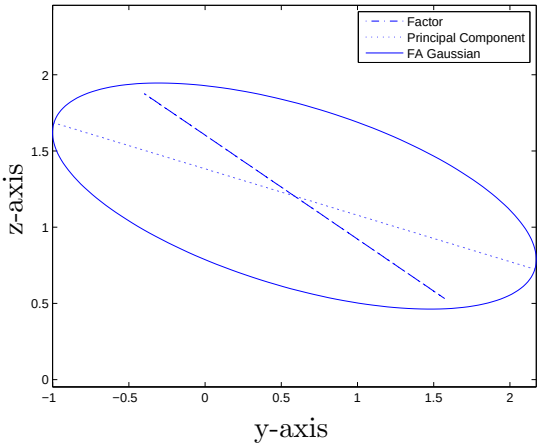


Figure 3.11: *FA covariance Gaussian representation of figure 3.9a with principal component and original factor.*

Now we can use these to estimate new values for $\mathbf{\Psi}$ and $\mathbf{W}$. This is the M-step:

$$\hat{\mathbf{W}} = \left(\sum_{i=1}^{N} \mathbf{v}_i \,\mathrm{E}(\mathbf{h}|\mathbf{v}_i)^T\right)\left(\sum_{l=1}^{N} \mathrm{E}(\mathbf{h}\mathbf{h}^T|\mathbf{v}_l)\right)^{-1} \tag{3.30}$$

$$\hat{\mathbf{\Psi}} = \frac{1}{N}\,\mathrm{diag}\left(\sum_{i=1}^{N} \mathbf{v}_i\mathbf{v}_i^T - \hat{\mathbf{W}}\,\mathrm{E}(\mathbf{h}|\mathbf{v}_i)\mathbf{v}_i^T\right) \tag{3.31}$$

where diag is the operator that extracts only the diagonal of its argument.

The equations 3.28 to 3.31 require multiple passes through the dataset $\mathbf{v}$. It would be much more efficient if we could replace the summations with matrix mathematics. We can replace summations over $\mathbf{v}\mathbf{v}^T$ with $\mathbf{\Sigma}(N-1)$ where $\mathbf{\Sigma}$ is the covariance matrix of $\mathbf{v}$.

We can now rewrite the equations 3.28 to 3.31 to the following:

E-step

$$\mathbf{E}_1 = \sum_{i=1}^{N} \mathbf{v}_i \,\mathrm{E}(\mathbf{h}|\mathbf{v}_i)^T = \sum_{i=1}^{N} \mathbf{v}_i\mathbf{v}_i^T\boldsymbol{\beta}^T = \mathbf{\Sigma}\boldsymbol{\beta}^T(N-1) \tag{3.32}$$

$$\begin{aligned}\mathbf{E}_2 &= \sum_{i=1}^{N} \mathrm{E}(\mathbf{h}\mathbf{h}^T|\mathbf{v}_i) = N(\mathbf{I}-\boldsymbol{\beta}\mathbf{W}) + \sum_{i=1}^{N} \boldsymbol{\beta}\mathbf{v}_i\mathbf{v}_i^T\boldsymbol{\beta}^T \\ &= N(\mathbf{I}-\boldsymbol{\beta}\mathbf{W}) + \boldsymbol{\beta}\mathbf{\Sigma}\boldsymbol{\beta}^T(N-1)\end{aligned} \tag{3.33}$$

M-step

$$\hat{\mathbf{W}} = \mathbf{E}_1\mathbf{E}_2^{-1} \tag{3.34}$$

$$\hat{\mathbf{\Psi}} = \frac{1}{N}\,\mathrm{diag}\left(\mathbf{\Sigma}(N-1) - \hat{\mathbf{W}}\mathbf{E}_1^T\right) \tag{3.35}$$

This being an instance of the EM algorithm, the E and M steps are repeated until the sum of all the terms in $\mathbf{E}_1$ stabilises.

To estimate the covariance from the above terms we calculate:

$$\hat{\boldsymbol{\Sigma}} = \mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi} \tag{3.36}$$

It must be remembered though, that the above estimated matrix is only valid when the assumption that the values of $\mathbf{v}$ has zero mean is true. When the mean is something other than zero one can estimate it from the data with equation 3.13. We then use $(\mathbf{v} - \boldsymbol{\mu})$ for the above estimations. The result of 3.36 will then give us the biased estimation of the covariance. To get the unbiased value we need to calculate:

$$\hat{\boldsymbol{\Sigma}} = \frac{N}{N-1}(\mathbf{W}\mathbf{W}^T + \boldsymbol{\Psi}) \tag{3.37}$$

This is due to the fact that the covariance is estimated using an already estimated value (the mean in this case) [22].

It must be noted that in the PCA- and PPCA methods, the $\mathbf{W}$ matrix consisted of a set of vectors that denoted the principal axes. With the FA method this matrix is the set of so-called *factor loadings*. There is no physical meaning to these factors and they are all considered equally likely as any rotated version of them. Also unlike the PCA- and PPCA methods, the columns of $\mathbf{W}$ need not be orthogonal to each other.

To get the inverse of the estimated covariance we again use the *Matrix Inversion Lemma*:

$$\hat{\boldsymbol{\Sigma}}^{-1} = \frac{N-1}{N}(\boldsymbol{\Psi}^{-1} - \boldsymbol{\Psi}^{-1}\mathbf{W}(\mathbf{I} + \mathbf{W}^T\boldsymbol{\Psi}^{-1}\mathbf{W})^{-1}\mathbf{W}^T\boldsymbol{\Psi}^{-1}) \tag{3.38}$$

This gives us a lower dimensional matrix inversion, which makes calculation faster. To calculate the log likelihood we use equation 3.16. We can use 3.38 to simplify this equation as we did for the PPCA case:

$$(\mathbf{x} - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) = \frac{N-1}{N}((\mathbf{x} - \boldsymbol{\mu})^T\boldsymbol{\Psi}^{-1}(\mathbf{x} - \boldsymbol{\mu}) - (\mathbf{x}')^T\mathbf{S}^{-1}\mathbf{x}') \tag{3.39}$$

with $\mathbf{x}' = \mathbf{W}^T\boldsymbol{\Psi}^{-1}(\mathbf{x} - \boldsymbol{\mu})$ and $\mathbf{S} = (\mathbf{I} + \mathbf{W}^T\boldsymbol{\Psi}^{-1}\mathbf{W})$. This is faster to calculate because there are less element multiplications. We can even further simplify this if we optimise the matrix multiplications involving $\boldsymbol{\Psi}$, because only the diagonal elements are non-zero.

3.5.2 Implementation

To estimate the FA covariance Gaussian PDF, we first need to estimate the full covariance Gaussian statistics from equations 3.13 and 3.14. Next we need to estimate the FA covariance matrix via the EM algorithm described in the previous section. Due to the fact that an EM algorithm is used to calculate an ML solution for the FA covariance matrix, this is *not* a closed loop solution. A predetermined number of iterations of the EM algorithm must be run, or until convergence takes place.

The fact that this method has no closed loop solution resulted in the fact that it is not very popular. In general, most of today's dimension reduction methods rely on the PCA, which is much easier to calculate and implement. It is also much faster!

Fortunately the ML training only takes place when the models are trained. When scoring, this model is still faster than the full covariance Gaussian, as can be seen from equations 3.16 and 3.39. We can also speed up scoring by storing the inverse of S in equation 3.39.

The likelihood from the scoring can be interpreted in the same way as the previous methods. The sum of the likelihoods of a set of vectors is equal to the total likelihood of the set. These likelihoods can again be compared to find the closest matching set.

It is possible to use the FA covariance Gaussian as a basis for a GMM. The E-step is calculated by equation 3.17. The calculation of the likelihood is obtained as described above. This is faster than calculation of the full covariance Gaussian likelihood. For the M-step we first calculate equations 3.18 to 3.20. Next we need to calculate the FA covariance matrix estimation from the full covariance matrix. This would usually mean calculating a number of EM algorithm iterations.

It would seem that this is computationally very expensive, as each step of the outer EM algorithm loop needs the calculation of a full EM algorithm for the FA covariance matrix. We can however simplify this by just doing one iteration of the ML estimation for the FA covariance matrix. With every outer loop iteration, the previous loop's calculated matrices are used as the initial conditions for the inner loop. One iteration of the inner loop is then calculated. In this way we do a simultaneous calculation of both EM algorithms. Practically it has been found that this method is a viable alternative and gives good results.

This combined EM-algorithm is calculated until the full system likelihood converges.

3.5.3 Strengths and Weaknesses

As before, we train a FA covariance GMM to fit to our original test set. This model, like all the models mentioned in this chapter, can hold a variable amount of correlation information. Depending on the number of factor loadings calculated, this method can at the least be equal to the diagonal covariance Gaussian (zero factor loadings) and at most to the full covariance Gaussian (when the factor loadings total one less than the number of dimensions).

This method should be faster than the full covariance Gaussian (depending on the number of factor loadings, as there is extra overhead in calculating the score with the FA parameters) and slower than the diagonal covariance Gaussian as there are more multiplications taking place during scoring.

For the following example we choose to calculate one factor loading. Taking any more would be as good as taking a full covariance matrix. We train twelve mixture components, the number that gave a good representation for a full covariance GMM. The results of this experiment are seen in figures 3.12 and 3.13.

The FA covariance GMM certainly does a better job than the diagonal and PPCA covariance GMMs with twelve mixture components. This is probably due to the way this method re-estimates the covariance. In this case, like the PPCA method, some direction's full covariance information is kept. These directions need not be in the directions of the axis as can be seen by figures 3.12 and 3.13.

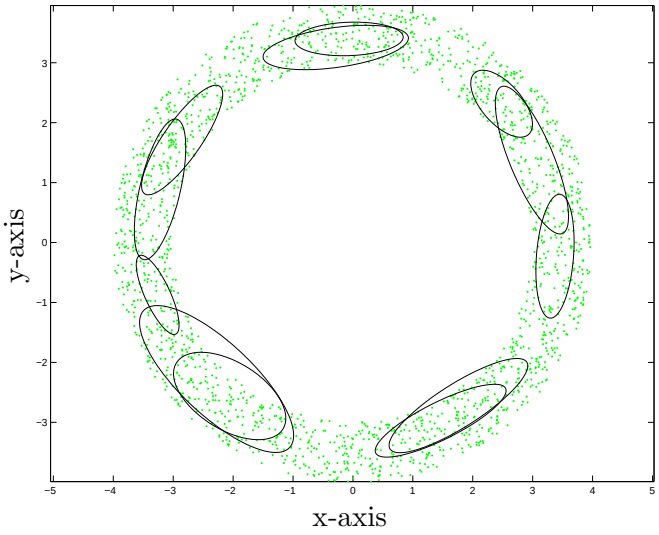


Figure 3.12: *Top view of a FA covariance GMM fit to test set (12 mixture components).*

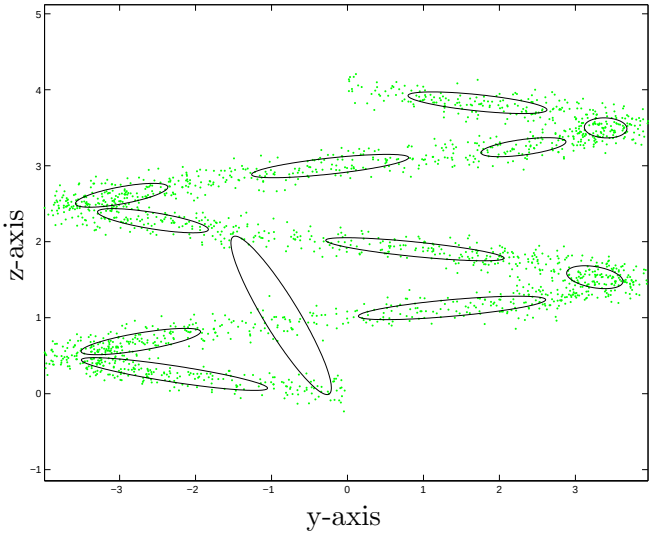


Figure 3.13: *Side view of a FA covariance GMM fit to test set (12 mixture components).*

The remaining directions orthogonal to these modelled directions are diagonally modelled. This eliminates one weakness of the PPCA method, where one could sometimes find a mixture component too 'wide' or 'narrow' because of the spherical generalisation. One can now see that most of the components fit well to the spread of the dataset in all dimensions.

Next we consider sixteen mixture components, a value that gave acceptable results for the diagonal covariance GMM and good results for both the sparse- and PPCA covariance GMMs. In figures 3.14 and 3.15 we can see that this method, as expected, gives a much

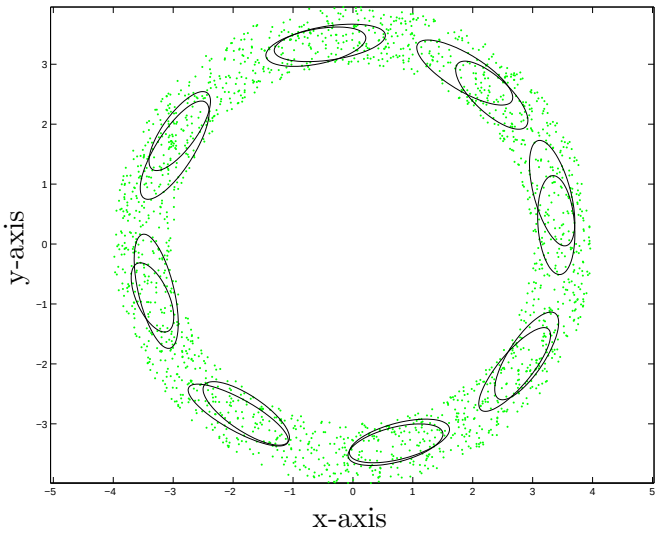


Figure 3.14: *Top view of a FA covariance GMM fit to test set (16 mixture components).*

better representation than the diagonal covariance GMM. It also gives a better representation than the sparse- and PPCA covariance GMMs. Even with fifteen mixture components this method covers the data better than the diagonal covariance method did with sixteen components. It is evident that this method is more accurate than the previous methods. There is a speed penalty however, though not as bad as the full covariance Gaussian.

Again, it is difficult to compare this method to the sparse covariance Gaussian method in a methodical way, as the methodologies are different. Comparing to the PPCA covariance method is easier, as they both are based on the same mathematical model. One can see that this method is more complex than the PPCA covariance method, due to the iterative estimation method and also due to the diagonal noise model.

This method can, as with the previous two methods, be scaled by adding or removing factor loadings. This range can reach from zero to as large as one would like to go, however, it would not make sense to take more factors than one less than the total dimensions, as this is enough to store the full covariance model. It follows that the higher the system dimensions, the wider the range of scaling. When the dimensions are higher, the gain in speed over the full covariance Gaussian is also more appreciated.

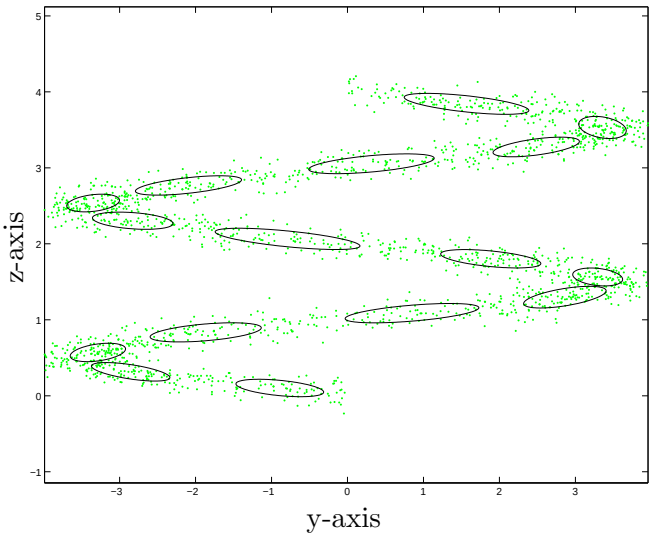


Figure 3.15: *Side view of a FA covariance GMM fit to test set (16 mixture components).*

This method needs less training data for a model than the full covariance Gaussian. It needs more data than the diagonal covariance Gaussian method, but only needs slightly more training data than the PPCA method, because like the PPCA method, only the S matrix needs to be inverted. The dimensions of this matrix are equal to the number of factor loadings calculated. Extra data is only needed to model the diagonal noise element. A strong point of this method is a much stronger model can be built when data is scarce.

The scalability of the FA covariance GMM via the adjustment of the mixture component number is dependent on the complexity of the components. Here the ratio of factor loadings per component to the number of components used can also be adjusted for best results. Over-training can also be avoided in this way.

For mixture components that have less than ideal training data that needs a more accurate model than the diagonal, sparse- and PPCA covariance models can give, this method is ideal, although it is slower.

This method, like the PPCA- and sparse covariance methods, can be used when one wants a more complex model than the diagonal covariance Gaussian without the penalties of the full covariance Gaussian. This method works better when mixture components of higher complexity is needed, as it models more information in its simplest form than the PPCA- and sparse covariance methods.

3.6 Summary

In this chapter we introduced three Gaussian PDF models that can be used in GMMs for ASR:

- The sparse covariance Gaussian PDF
- The PPCA covariance Gaussian PDF
- The FA covariance Gaussian PDF

We first discussed a general linear transform used for dimension reduction and used the PCA and LDA as examples of dimensionality reduction. This was done as two of the above methods (the PPCA- and FA methods) are based on the general transform.

Next we discussed each method in detail concerning the theory, implementation and the strengths and weaknesses. We included a small demonstration of each method's capabilities.

For the sparse covariance Gaussian PDF we found:

* It is similar to the diagonal covariance Gaussian PDF, but keeps the information regarding certain correlations between dimensions
* It is complex to implement the algorithm, but the algorithm is non-iterative
* The training is more expensive than the diagonal covariance Gaussian, as the diagonal plus the sparse values are estimated
* The number of mixture components needed to model the data depends on the chosen model complexity
* Scalability of the GMM via the number of mixture components is dependant on the complexity of the components
* Over training can be countered by decreasing the number of mixture components or by decreasing the component complexity
* The complexity of the mixture components can be set to best fit to the amount of training data available

For the PPCA covariance Gaussian PDF we found:

* It is based on the PCA and is a PPCA reduction of the full covariance Gaussian PDF
* It is easy to implement and the algorithm is non-iterative
* The training is more expensive than the diagonal covariance Gaussian, because eigenvalue decomposition is needed for a full covariance matrix
* The number of mixture components needed for a useful model is dependant on the model complexity
* Scalability of the GMM via adjustment of the mixture component number is dependant on the complexity of the components

* Over training can be avoided by using less complex mixture components, or reducing the number of components

* If the training data is scarce, the mixture component complexity can be decreased

For the FA covariance Gaussian PDF we found:

* It is based on the FA transform and is a FA reduction of the full covariance Gaussian PDF

* It is more complex to implement than the PPCA covariance method and the resultant algorithm is an ML estimation that is iterative, although when used in a GMM this can be integrated in the GMM EM steps

* The training is more complex than the diagonal covariance Gaussian because both the factors and the diagonal of the noise element needs to be estimated

* The number of mixture components needed to train a representative model is dependant on the complexity of the components

* The scalability of an FA covariance GMM via adjustment of mixture component number is dependant on the complexity of the components

* Over training is avoided by minimising the mixture components complexity or decreasing the number of components

* The model complexity can be minimised if the training data is scarce

In this chapter we see that the sparse- and PPCA covariance PDFs seem similar in complexity, although it is difficult to compare as their methodologies differ. Both methods can be chosen when one needs a better model than the diagonal covariance Gaussian, but the full covariance Gaussian is too complex. Both models are scalable via model complexity. The sparse covariance Gaussian is scalable between the diagonal and full covariance representations, and the PPCA covariance Gaussian is scalable between the spherical and full covariance representations.

The FA covariance PDF is more complex than the PPCA covariance PDF, as more information is modelled with the noise element. It is slightly more expensive, but if more accuracy is required, it should be faster to use FA covariance PDFs with a small number of factor loadings than PPCA covariance PDFs with a higher number of principal components, as higher PPCA mixture components would require more unit multiplications. It is also ideal when data is scarce and a stronger model is needed. Its components can also be scaled, in this case between the diagonal and full covariance representations.

Chapter 4

Test System Implementation

4.1 Introduction

In this chapter we will familiarise the reader with the test system used to evaluate the methods discussed in chapters 2 and 3. There are two main parts to the test system implemented to investigate the various methods.

Each part implemented is coupled with a speech corpus. The complexity of each part is due to the size of each corpus and the amount of information included in the transcriptions.

Each part is discussed in isolation. First we discuss the corpus used. Each step, from feature extraction to the final transcription is discussed. The baseline systems are described along with their variations.

4.2 Test System: Initial Model Training

4.2.1 The NTIMIT speech corpus

This part is trained with the NTIMIT speech corpus. The NTIMIT dataset is a telephone bandwidth version of the TIMIT speech corpus. This dataset consists of speech that is arranged in eight groups, each one donating a different English dialect. In total it has 630 speakers, of which 192 are female and 438 are male.

The audio was recorded at Texas Instruments (TI) and transcribed at the Massachusetts Institute of Technology (MIT). The original TIMIT speech was sampled at 16kHz using 16 bits/sample. The time durations for all the phonemes are included in the transcriptions and are assumed to be correct.

For the NTIMIT application the TIMIT speech corpus was transmitted over a telephone network. A more comprehensive description of this corpus can be found in Appendix A.

4.2.2 Signal Processing

Preprocessing

Before feature extraction we do some preprocessing on the speech signals. First we do preemphasis [21]. Preemphasis involves filtering the speech signal in such a way that the higher frequencies are emphasised.

The reason for this is because in speech the lower frequencies usually contain more energy. When trying to model the speech signal, we want to have a good model of the higher frequencies as well as the lower ones. Without preemphasis the models might favour the lower frequencies above the higher frequencies.

Next we want to look at the average power of the speech signals. Because of changing conditions when the speech is recorded, some signals have more power than others. This will create bias towards signals with higher power when we do recognition. We thus calculate the power for each signal and scale each one to have unity power.

Feature extraction

Using the speech signal as a whole for speech recognition is difficult, as it would lead to very high dimensional vectors and the dimension number per vector would differ, making comparison difficult.

A way to extract a predetermined number of dimensions from the data is to use cepstral coefficients [3, 5, 21]. We use the Discrete Cosine Transform (DCT) with a Mel-scale filter bank to get the Mel-scale Frequency Cepstral Coefficients (MFCCs). The MFCC values have proven to be one of the best methods of extracting speech information from the raw data. We choose to use a Mel-scale filter bank with eighteen filters. We take speech blocks of 0.03 seconds and advance the block at 0.015 seconds at a time. We then take the first twelve cepstral coefficients of each block.

Post-Processing

One of the problems of recording speech with a microphone, is the fact that no two recordings have the same characteristics. The background noise may differ and even things such as distance from microphone or movement during speech makes a difference. This difference usually manifests itself as noise convoluted with the speech signal [16]. This is also referred to as channel effects.

This translates into an addition to the cepstral coefficients. This addition gives the signal a bias. A simple way to remove this bias is to subtract the mean of each dimension from the cepstral coefficients. This is called Cepstral Mean Subtraction (CMS).

Because we use a computer to calculate the covariance matrices and computers have precision issues when using real (i.e. not integer) values, we want to scale the variances of the cepstra in such a way that these precision errors are minimised. For each set of cepstra

we calculate the variance for each dimension and scale it so the variance over that dimension is equal to unity.

The traditional next step is to try and capture additional information in terms of the rate of change of the cepstral coefficients (otherwise known as the *temporal* information). The derivatives of the cepstral coefficients are calculated and appended to the vector (essentially doubling the dimension). These derivatives are known as the *delta coefficients*. These are usually calculated by polynomials and include the previous two and following two feature vectors to get a stable value.

The derivative of the derivative is also calculated and appended to the vector (now making it triple its original dimension). Afterwards some type of linear transform (usually LDA) is applied to find the best low dimensional representation of the feature vectors.

These delta coefficients are nothing more than another type of linear transform. With the δ and $\delta\delta$ calculations, each vector has a connection to four previous feature vectors and four following feature vectors. Instead of calculating the derivatives, we can just replace each vector with a concatenated version of the four previous feature vectors, itself and the four following feature vectors. This contains all the information used as input for the δ and $\delta\delta$ calculations.

The final step is to use a linear transform to do dimension reduction (see section 3.2). We use the LDA method, because it ensures maximum separation between classes [7, 25]. We reduce the dimension to a number that still holds most of the information, but is more manageable to use for calculations.

If either the delta coefficients are added, or a set of feature vectors are concatenated, the LDA is nothing more than a method to find the optimal linear transform that reduces the dimensionality of the data in the feature vectors to ensure maximum class separation. When the δ and $\delta\delta$ coefficients are calculated, the information in nine feature vectors are reduced to these two sets of coefficients. This might not be the optimum linear transform.

When we do concatenate the four previous feature vectors, the current vector and the four following feature vectors and supply this directly to the LDA, we are ensured of the optimal linear transform for class separability. Informal testing has shown a small decrease in error rates if this vector is used instead of the δ and $\delta\delta$ coefficients.

With the LDA, twenty-four is chosen as the final number of dimensions for our system as the eigenvalues show that more than 95% of the information is still retained.

4.2.3 Pattern Recognition

GMM Training

In the transcriptions we have information telling us which segments of the speech data are linked to which phonemes. The first thing we do is to subdivide the phonemes into sets of three or four (depending on the length of the phonemes). We include a silence model that is not subdivided. We also include an 'other' model that is not subdivided. The 'other' model

contains all speech data that is not defined in our set of phonemes.

Where we do divide the phonemes, the initial choice is to make the subsegments equally long. We then group all the data found in each subsegment together with other instances of this subsegment. We then train a GMM (discussed in chapters 2 and 3) for each subsegment of each phoneme. For our system we use the Tree-based Adaptive Gaussian Mixture Model (T-BAGMM) [4].

This method assumes that the mixture components of a GMM are the leaf nodes of a binary tree structure. Before the data is trained, a maximum leaf node number is given. For the first baseline set we select the value to be twenty-four. Now the training initially begins with one (root) node. Every second training iteration, each node is evaluated for a split using eigen-analysis and using this information to split the node along its principal axis into two equal nodes.

If a node is found to not significantly contribute to the GMM score, it is not split. The number of feature vectors in the nodes are limited to a minimum of ninety-nine. This number is chosen as the minimum number of features we want in a PDF to ensure a good statistical representation of the feature set. If there are less than ninety-nine feature vectors, the node and its sibling (the other leaf of its parent node) are discarded and the parent node will be evaluated instead. This is to avoid under-trained mixture components.

The final GMM will then have anything between one and twenty-four leaf nodes, depending on the final outcome of this training method. These leaf nodes are used as the GMM mixture components.

We use this method for two reasons:

1. Because the number of leaf nodes change dynamically during training, it is more likely that a number of mixture components that ensure a better model will be the result, rather than having to choose a fixed component number initially.

2. This method yields a significant increase in training and testing speed over the standard GMM, as mixture components that do not significantly contribute to the GMM score, are evaluated at a lower complexity level.

HMM Training

One of the disadvantages of the GMM is the fact that no time information can be stored in this structure. All the feature vectors in a GMM are evaluated as a static set. When evaluating speech, time information is critical. Research shows that most phonemes do not sound the same at its beginning than at its end. It would be favourable to be able to associate the feature vector with the specific time frame. This is where the Hidden Markov Model (HMM) is useful.

The HMM is essentially a set of states connected by transition probabilities [23, 24]. Each state has its own PDF distribution that describes the data that is likely to be present when that state is reached. This PDF distribution can be anything from a single PDF, a GMM

or even another HMM. The HMM is of such a nature, that different sequences of the same feature vectors will give a different result. This is different from a GMM or any stand-alone PDF.

We set up the HMMs in left-to-right format. This means that each state only has two transitions. One to itself and one to the state to its right. This models a phoneme in parts. For each phoneme that was split into three subsegments, a three-state left-to-right HMM is used (see figure 4.1). The three GMMs modelling the subsegments are put in the states from left to right, in chronological order. For the phonemes divided into four parts, the same is done with four-state left-to-right HMMs.

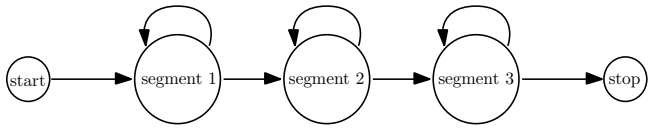


Figure 4.1: *A three-state left-to-right HMM.*

Now we train the HMMs by using the full phonemes (not subdivided) found in the transcriptions. Each HMM now represents a phoneme. The full set of feature vectors for the phonemes are now used to train the HMMs. The training algorithm used is based on the Viterbi algorithm [2, 6].

The Viterbi algorithm gives us the path through the HMM with the highest probability for each feature vector set. For all the feature vector sets, the feature vectors are then grouped according to the states they visited in this path. Also all the transitions from each state are counted. This information is used to reestimate the PDFs in the states and also the new transition probabilities. This is another instance of an EM algorithm. It is iteratively run until the parameters stabilise.

Forced Alignment

After we have the trained HMM models, we address the assumption we made earlier that the phonemes are phonetically divided into equal segments. We assumed that phonetically the transitions within a phoneme occurs in equal lengths. This is unrealistic and here we use forced alignment to redefine these segment boundaries.

The original transcribed data has fixed boundaries at the beginning and end of each phoneme. As the name suggests, these boundaries stay fixed. It is only the subdivided segments we are interested in. We use the newly trained phoneme HMMs for each phoneme to do the alignment. For each speech segment, the phonemes are extracted according to the fixed boundaries. The Viterbi algorithm is run for each phoneme with its corresponding HMM. When the Viterbi path is calculated, it is used to realign the segment boundaries. Because we know that the feature vectors are calculated in 0.015 second increments, we can calculate the duration of each segment. This new duration is written to file to create the new transcriptions.

One must note, that because the boundaries have changed, we have to retrain the LDA (see section 4.2.2). This is because the definition of our classes have changed and the LDA ensures maximum class separation.

Retrain GMMs and HMMs

Now we retrain the initial GMMs with the new aligned data as we did in earlier in this section. We also initialise the HMMs the same way as before.

We should get better HMM models of our phonemes, as the GMMs used in the states of the HMMs will now be more representative of the transitions inside the phonemes. The forced alignment procedure can be repeated until the results converges (almost like an EM algorithm), but it has been found that more than one forced alignment usually has no extra benefit to the accuracy.

4.3 Test System: Final Model Training

4.3.1 The AST Speech Corpus

The second part is trained on the African Speech Technology (AST) speech corpus. Specifically the English database is used. This database consists of five variants of South African English dialects. This includes Standard English, Black English, Coloured English, Asian English and Afrikaans English. All these dialects were used in this test system, excluding the Black English.

The speech in the AST corpus was recorded from the South African telephone network and contains a mix of telephone and cellphone recordings. Each dialect contains approximately between 300 and 400 calls and each call contains from a few to 40 utterances [26]. The speech is transcribed phonetically using a phoneme set similar to the NTIMIT set. The transcriptions are not time aligned. Thus we know the order in which the phonemes occurred, but not the time.

4.3.2 Signal Processing

The whole signal processing process, from preprocessing to post-processing is identical to the NTIMIT part of the system as described in section 4.2.2. As this corpus also contains telephone data, the feature vectors can be calculated in the same way.

4.3.3 Pattern Recognition

Here we deviate from the NTIMIT training methods. The main reason is the fact that the transcriptions are not time aligned. Previously we would determine which feature vectors belonged to which phoneme and train the GMMs. Now, however, there is no direct way to

determine which feature vector belongs to which phoneme. This complicates the training process. We do the training in three steps.

Training First Order HMMs

Because the NTIMIT and AST signal processing is similar, it is reasonable to assume the phoneme models would be similar. To enforce this assumption, we use the same post-processing as we did on the NTIMIT feature vectors for these models instead of calculating new ones.

It is unrealistic to assume that these models would give good results when used for scoring, but it is useful to use as initial models for the training process. We use these models as initial HMMs and train them to the feature vectors calculated from this corpus.

The HMM training technique also differs here from the one in section 4.2.3. Because we still do not know the exact time for each phoneme, we use a technique called embedded training.

With this technique we train on the feature vectors from a full speech file. We know the order in which the phonemes occur, thus we can create a many-state HMM that is constructed by concatenating all the phoneme HMMs from left to right in the order which they occur in the feature vector set (see figure 4.2). With every iteration of the Viterbi training algorithm

Figure 4.2: *An HMM consisting of a set of phonemes occurring in the utterance.*

the feature vectors are aligned to the phoneme HMM states according to the best path through the HMMs. Note that embedded training does not alter the transcriptions.

Next we want to rebuild the models using the data from this system alone. To rebuild the models we need time aligned transcriptions. This can be done with forced alignment. To ensure that our alignment is accurate, we want to make our models as accurate as possible. For forced alignment, training time is not as big an issue compared to evaluation time and accuracy. We cannot adjust the complexity of the GMMs at this stage, but we can do something about the HMMs.

Training Second Order HMMs

One way to make the HMMs more accurate is to use a second order HMM for the phoneme modelling instead of first order HMMs. With a first order HMM when jumping from one state to the next, only the current state information influences the transition probability. With the second order HMM, the transition probabilities are also affected by the state visited at the previous time instance. Thus the probability of a transition is determined by both the current and the previous state. Figure 4.3 illustrates a second order HMM.

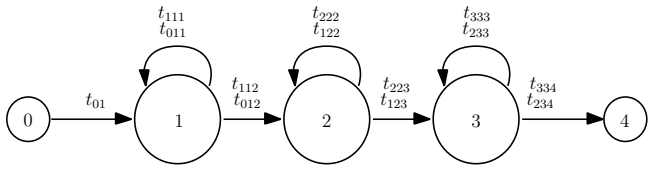


Figure 4.3: *Example of a second order HMM.*

When training a second order HMM, each transition needs to have an outcome based on all the possible previous states. This increases the complexity of the training algorithm. The added complexity means training a second order HMM with the Viterbi algorithm is unrealistic to implement. However, there is a method that can be used to reduce an n^{th} order HMM to an $(n-1)^{th}$ order HMM. This is the Oder Reducing algorithm (ORED) introduced in [9].

When we reduce the second order HMM in figure 4.3, we get the first order equivalent in figure 4.4.

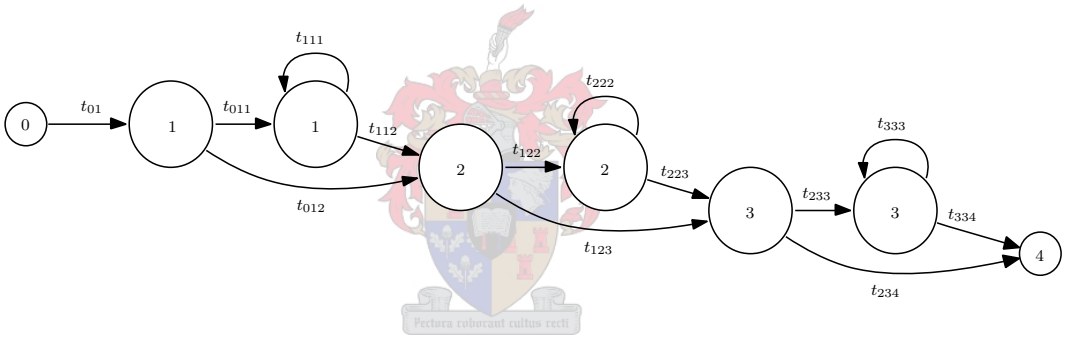


Figure 4.4: *A first order reduced HMM from the second order HMM in figure 4.3.*

The indices in the states of figure 4.4 refer to the states of figure 4.3. We see some repetitions in the reduced version. This means that the PDF is shared by these states. These states can be initialised by training the HMM in figure 4.3 as a first order HMM. This is something we have already done in the first training step. The extra transitions can also be initialised to the values of the lower order transitions they are based on.

The phoneme HMMs are defined as second order and transformed to their first order equivalents. They are initialised by the GMMs and transitions from training step one are retrained. Embedded training is used with these new HMMs. The resultant trained HMMs should model the phonemes better due to the extra information included in the model.

Retraining First Order HMMs

The new second order HMM equivalents are used to force-align the transcriptions as described in section 4.2.3. These forced aligned transcriptions are then used to train a new

LDA, because the definition of the phoneme segments have changed. The resultant feature vectors from this new linear transform are then used to train new GMMs for the HMM states. The forced aligned transcriptions now give us time aligned phoneme segments we can use. The number of mixture components used to train the phoneme segments for our first test is twenty-four.

After these GMMs are trained, they are used to initialise the states of the original first order HMMs used to model the phonemes (as in section 4.2.3). These new phoneme HMMs are then trained using embedded training (using the original transcriptions again, not the force aligned ones). These phoneme HMMs are the final models for this test system. The reason they are not second order equivalents is because we want to maximise the transcription (i.e. scoring) speed.

4.3.4 Testing

For test results we want to use the final trained phoneme HMMs to transcribe the test data. The test data consisted of a separate test set A The resultant transcriptions are then compared to the (assumed) correct transcriptions. A percentage of correct phoneme placings is calculated (taking into account deletions and substitutions). A final error rate is calculated after penalising for phoneme insertions.

To transcribe the data, a phoneme spotter is constructed. The phoneme spotter starts as single state HMM (seen in figure 4.5). This HMM is combined with a parallel HMM, having

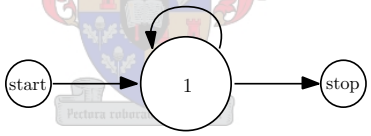


Figure 4.5: *A single state HMM.*

as many parallel states as there are phonemes. Figure 4.6 illustrates a three-state parallel HMM. The combination of figures 4.5 and 4.6 gives the HMM in figure 4.7. In this figure

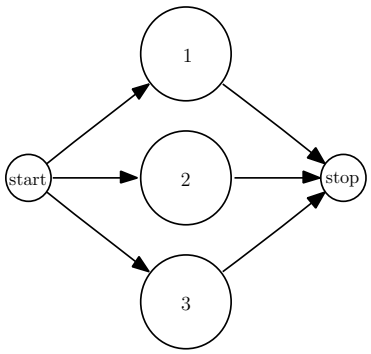


Figure 4.6: *A three-state parallel HMM.*

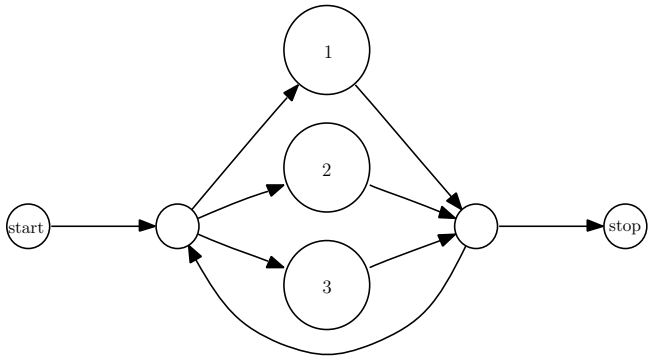


Figure 4.7: *A combination of the single state and parallel HMMs.*

we can see that states 1, 2 and 3 are the states of the same numbers from figure 4.6. These three states replace the one state from figure 4.5. The empty states are only transition states and contain no PDFs. They are there to facilitate the loop back from figure 4.5.

The reason for this is because states 1 to 3 are set to contain the phonemes. When we want to transcribe piece of speech, we must be able to return to the beginning for the next phoneme. In our spotter there are as many states in the parallel HMM as there are phonemes. Next we put each phoneme HMM into each of these states.

We now take each test speech file and do the same preprocessing, feature extraction and post-processing described in section 4.2.2. The Viterbi algorithm is then run for the spotter with each set of test feature vectors as input. When the best path is calculated, each state/HMM visited is recorded to file with a time-stamp. This is the newly generated transcription.

These transcriptions are then compared to the (assumed) correct transcriptions as mentioned earlier in this section. The resulting error rate is the metric used.

4.4 Baseline Systems

The baseline systems are the standard and all comparisons are made in relation to these systems. The diagonal covariance Gaussian will be used for the mixture components in these systems. This decision is made because the diagonal covariance Gaussian is the preferred PDF in ASR systems. For each of the tests, the component PDFs will be replaced by one of the models described in this thesis.

Three baseline systems are trained. One with a target of twenty-four mixture components and on the full training data, one with a target of twenty-four components on a reduced training set and one with a target of three components for evaluations on smaller mixture models. The error rates for the baseline systems are calculated as described in section 4.3.4. These results can be seen in Table 4.1.

Baseline System	Error Rate
Twenty-four components	49%
Twenty-four components (scarce training)	52%
Three components	57%

Table 4.1: *Error rates for the baseline systems. These systems all use the diagonal Gaussian covariance.*

4.5 Summary

In this chapter we discussed the test system to be used to evaluate the PDF methods discussed in this thesis. The system used for this evaluation uses two different speech databases:

1. The NTIMIT speech corpus
2. The AST speech corpus

We discussed issues regarding

- Signal Processing,
- Pattern Recognition and
- Testing

regarding each database.

Topics that were discussed in relation to signal processing were:

Preprocessing: We discussed methods such as *preemphasis* to emphasise the higher frequency components and *power normalisation* to make all signals have equal average power.

Feature Extraction: Here we discussed the use of *Mell-scale Frequency Cepstral Coefficients (MFCCs)* to extract speech information of all signals into a similar comparable form.

Post-Processing: Here we discussed using *Cepstral Mean Subtraction (CMS)* to reduce the bias contributed by channel effects and background noise. We used *variance normalisation* to make the covariance matrices more robust for computer calculation. We also discussed extending the feature vectors to include temporal information by either using *delta-* and *delta-delta coefficients* or to append a set of preceding and a set of following feature vectors to the current feature vector. *Dimension reduction* was also discussed, which is achieved by using *Linear Discriminant Analysis (LDA).*

Topics discussed in relation to Pattern Recognition (PR) were:

GMM training: We discussed the sectioning of the phonemes and training GMMs with these sections using the *T-BAGMM* algorithm.

HMM training: We discussed the form and initialisation of the HMM models for the phonemes. We also discussed the method of training using the *Viterbi* algorithm.

Forced Alignment: We discussed how we used the trained phoneme HMMs to recalculate the boundaries of the sections of phonemes. These new sections were used to retrain the system from scratch.

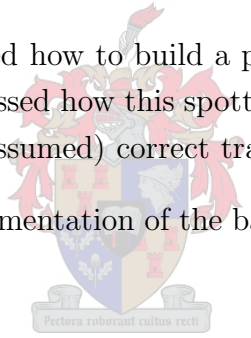
Embedded Training: We discussed how to use phoneme HMMs to train similar data where the phoneme boundaries are unknown.

Second Order HMMs: We discussed the significance of using second order HMMs instead of first order HMMs for modelling. We also mentioned the *Order Reducing (ORED)* algorithm for reducing a second order HMM to a first order equivalent and the training thereof.

In relation to Testing we discussed:

Phoneme Spotter: We discussed how to build a phoneme spotter using the trained phoneme HMMs. We also discussed how this spotter can transcribe the data and how this data is evaluated against (assumed) correct transcriptions.

Finally we looked at the implementation of the baseline systems and the error rates they achieve.



Chapter 5

Experimental Results

5.1 Introduction

In this chapter we consider various tests using the baseline systems discussed in chapter 4. We first do tests comparing the PDFs defined in chapters 2 and 3. We train equivalent systems with each and observe the differences in their error rates.

Next we train systems for each PDF type that have roughly the same accuracy as the baseline system by altering the mixture components. We compare system with a target of twenty-four components with an equivalent system using less training data. We also investigate a system with a very low number of mixture components. The evaluation speeds are then compared to determine the best system for the given situation.

The results for each test are discussed for each system compared to the baseline and compared to each other. Finally we summarise all the test results and discuss their significance.

5.2 Comparative Testing

As described in chapter 4, the baseline systems used for these tests have mixture components of diagonal covariance PDFs. First we set a target of twenty-four components in every system and compared the results and the time scoring takes. This is to highlight the differences for each system.

Each mixture component has to contain more than ninety-nine samples to prevent components that are too scarcely modelled. We tested the following PDFs:

1. A spherical covariance Gaussian
2. A full covariance Gaussian
3. A sparse covariance Gaussian with a block size of two and a minimum correlation coefficient of 0.4
4. A PPCA covariance Gaussian with at most one principal component and a minimum eigenvalue coefficient of 0.083

5. A FA covariance Gaussian with one factor loading

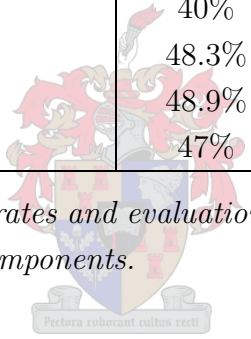
We choose the sparse covariance parameters so that the system is still close to diagonal. This way one can see the effect of a small percentage of added information over the baseline system. The same reasoning is used for the PPCA- and FA methods. The smallest increment of extra information is taken.

With the PPCA method we used the value of 0.083 for the minimum eigenvalue parameter (see section 3.4). As stated, for a mixture component approaching a spherical covariance Gaussian the eigenvalues are similar, giving a twenty-four dimensional Gaussian eigenvalues of 0.042 of the sum of all the eigenvalues. We again take twice this value as the minimum value to account for some variation.

These systems are then compared to the baseline system. We can see these results in Table 5.1 (with the evaluation time as a percentage of the baseline system).

PDF (Gaussian) Type	Error Rate	Evaluation time
Diagonal (Baseline)	49%	100%
Spherical	55.7%	69.4%
Full	40%	455.8%
Sparse	48.3%	134.8%
PPCA	48.9%	169.6%
FA	47%	177.9%

Table 5.1: *Comparison of error rates and evaluation speed for differing PDF-types with the same target number of mixture components.*



Let us first look at each PDF individually:

The Spherical Covariance Gaussian PDF

We can see that the spherical covariance Gaussian does indeed do worse than the baseline system. This is intuitive, because we generalise each mixture component with only one parameter. We have a relative 13.7% increase in error rate. With a 30.6% decrease in evaluation speed, we can see this method is faster to score against.

The Full Covariance Gaussian PDF

This method, as stated in chapter 2, models the maximum covariance information per mixture component. This comes at a severe speed penalty. We get a relative 18.4% decrease in the error rate over the baseline system with a 355.8% increase in evaluation speed. This system is the most accurate, but also the slowest of all the systems here.

The Sparse Covariance Gaussian PDF

The sparse covariance Gaussian seems slightly more accurate than the baseline system with a relative 1.4% improvement in error rate. We can see by just adding a small amount of extra information (as is the case with the maximum block size at 2 and the minimum correlation coefficient value at 0.4), we get an improvement in the error rate over the baseline system.

We do pay a price in evaluation speed. This method is 34.8% slower during evaluation.

The PPCA Covariance Gaussian PDF

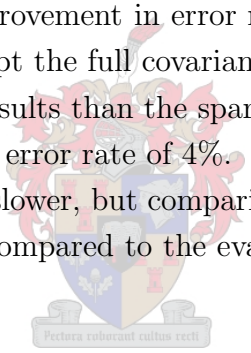
We can see that when only taking one principal component, the PPCA Gaussian is very close to the baseline system in accuracy, with the relative difference in the error rate being 0.2%.

With the evaluation this method is 69.6% slower than the baseline system. It would seem that this method is more expensive in time, with no real improvement over accuracy.

The FA Covariance Gaussian PDF

This method shows a greater improvement in error rates to the baseline system when compared to the other methods (except the full covariance Gaussian). When one factor loading is used, this model gives better results than the sparse- and PPCA covariance test systems. We get a relative improvement in error rate of 4%.

The evaluation time is 77.9% slower, but comparing this to the PPCA method, the error rate is significantly better when compared to the evaluation time.



Overview

When one looks at all the results, we see that although the full covariance Gaussian is the most accurate, it is very slow and not practical to implement for these systems.

Next we have the FA covariance Gaussian PDF. With only one factor loading it gives better results than the baseline. While it is only faster than the full covariance Gaussian, the error rate improvement is the most significant of all the generalised PDFs.

The sparse covariance Gaussian also shows some improvement over the baseline system. It is less accurate in this form than the FA covariance Gaussian, but faster to evaluate.

Considering how close the PPCA covariance Gaussian is related to the FA covariance Gaussian (only the noise elements differ), it is surprising to see the difference in accuracy. It is less accurate by quite a margin and is almost at the same error rate as the baseline system. It is somewhat slower than the baseline and the longer evaluation time seems unjustified if compared to the other methods here.

The spherical Gaussian is the least accurate of the group. This is expected and as previously mentioned, this method needs a high number of mixture components to accurately model data. The more components used, the higher the EM algorithm overhead gets. This method is thus also not very practical for this type of implementation.

It is clear that the methods compared with each other differ in their own ways. The diagonal and spherical covariance methods differ in the assumptions made on the data being modelled and the sparse-, PPCA- and FA covariance methods differ in the decisions made on what data to regard as important and what to generalise. Each shows different accuracy.

5.3 Evaluation Speed Tests

While accuracy testing is important, another important factor is the speed of the system. If the PDF model is very accurate, but takes too long to score (as with the full covariance Gaussian), it is impractical to implement.

5.3.1 First Baseline: Twenty-four Mixture Components with a Full Training Set

First we alter the mixture components of each method to get close to the error rate of the twenty-four component diagonal covariance Gaussian baseline system. We take the same set of PDFs tested in the previous section. We exclude the spherical covariance Gaussian, as this system could not get close to the baseline error rate even with a high number of components. With a very high number of components the number of samples per component becomes less than the ninety-nine sample limit (see chapter 4).

With the baseline having an error rate of 49%, we scaled each method to have an error rate of between 48% and 49.6%. This gives a relative difference of 1.2–2%.

The speed of the baseline system is taken as reference and the rest are calculated as a percentage of the baseline system. We do the same for the number of parameters for each PDF. We also calculate the approximate total number of mixture components in each system. The results of this test can be seen in table 5.2.

PDF (Gaussian) Type	Components	Parameters	Error Rate	Evaluation Speed
Diagonal (Baseline)	2945	100%	49%	100%
Full	340	77.9%	48.6%	242.9%
Sparse	2353	79.9% - 99.9%	49.6%	128.9%
PPCA	2843	50.3% - 98.5%	48.8%	168.7%
FA	1985	101.1%	48%	161.9%

Table 5.2: *Comparison of different PDFs to the first baseline. Note the Sparse and PPCA methods have a minimum and maximum range of parameters due to the way each mixture component is evaluated.*

We again first look at all the results in isolation:

The Full Covariance Gaussian PDF

To get close to the baseline accuracy, we only need about 340 mixture components for the full covariance Gaussian (with a 48.6% error rate). Each component contains a symmetrical twenty-four dimensional covariance matrix and a mean vector. This equates to 324 parameters per component. Compared to the baseline that has 48 parameters per component, we see a 22% reduction in parameters.

Even with the smaller number of parameters, this system takes 142.9% longer to evaluate than the twenty-four mixture component baseline. This is due to the overhead of doing full matrix multiplication instead of the vector multiplication for the baseline. It shows that the inclusion of the off diagonal elements increases the mathematical complexity, even when the total number of parameters is less than the baseline.

The Sparse Covariance Gaussian PDF

This method needs 2353 mixture components to compete with the baseline accuracy (a 49.6% error rate). Each component contains a mean vector, a diagonal covariance vector, and between zero to a maximum of twelve blocks of a size of 2. The number of blocks depend on how many correlation coefficients are larger than the minimum correlation parameter. Each block has one covariance parameter. Although the indices of each block member are also stored, the information is only used to store the position of each parameter on the sparse covariance matrix. This means each mixture component has between 48 and 60 parameters (excluding the indices of correlations). Therefore the system has up to 20.1% less parameters than the baseline.

This system takes 28.9% longer than the baseline system to evaluate. Having possibly less parameters makes it more complex per parameter to evaluate. The larger the block size, the more parameters are stored.

The PPCA Covariance Gaussian PDF

For this method, we need 2843 mixture components for similar accuracy to the baseline (48.8% error rate). Each component contains a mean vector and at least one noise parameter (when the minimum eigenvalue parameter rejects the eigenvalue) and a maximum of one principal component with a noise parameter. Each component has either 25 or 49 parameters. This system has therefor possibly between 1.5–49.7% less parameters than the baseline.

This system takes 68.7% longer to evaluate than the baseline. It would seem that even considering the possible saving in parameters, this system is mathematically more complex. Even with a one dimensional inversion it is much slower. For a higher number of principal components, this method would become more expensive due to a larger number of parameter multiplications.

The FA Covariance Gaussian PDF

This method only needs 1985 mixture components to have similar accuracy to the baseline (a 48% error rate). Each component contains a mean vector, a factor vector and a noise element vector. Each component therefor contains 72 parameters. This system has 1.1% more parameters than the baseline.

This system takes 62.9% longer to evaluate than the baseline. While it has almost the same number of parameters, using the parameters to calculate the scores are more complex and thus slower. The more factors we include, the slower this method becomes due to more parameter multiplications.

It is interesting to note that this method is faster than the 'simpler' PPCA method. More accuracy is gained by the better noise model than time is lost to evaluate it.

Overview

These results show that the baseline system is the fastest here by at least 28.9%, yet it needs more mixture components than the other methods to reach the same error rate.

Next we have the sparse covariance Gaussian system. This system is slower to evaluate by almost 30% even though we need less components than the baseline and possibly have less parameters.

The FA covariance Gaussian follows. This system is 62.9% slower than the baseline system to evaluate. It needs less mixture components than the sparse covariance Gaussian and has slightly more parameters than the baseline system.

The PPCA covariance Gaussian method is 68.7% slower to evaluate than the baseline system. It has slightly less mixture components than the baseline and could have a variable number of parameters ranging from about half the baseline to about the same. The saving in parameters does not make up for the slower evaluation speed.

The full covariance Gaussian system is 142.9% slower to evaluate and has the least number of mixture components. Although it has less parameters to evaluate, the full covariance Gaussian is very slow to evaluate and does not improve on the baseline system.

From the collective results, the diagonal covariance Gaussian system is the clear winner here. The gain in speed over the other methods range between 28.9–142.9%. These results indicate that even if the correlation parameters do contain extra useful information, the added complexity in evaluation for these parameters far outweigh their gain in accuracy. It seems that, if possible, it would be less time consuming to increase the number of mixture components of a diagonal covariance Gaussian system than to explicitly try to model the correlations when higher accuracy is needed.

5.3.2 Second Baseline: Twenty-four Mixture Components with a Reduced Training Set

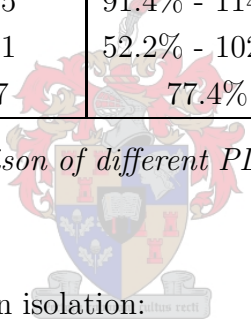
Next we take the same configuration than with the first baseline system, but decrease the training set to about a sixteenth of the full set. The ratio of speakers are kept the same. We then train the baseline system on this training set. Next we again train systems with each method varying the mixture components to get close to the baseline accuracy. Again we excluded the spherical covariance Gaussian as we could not reach the desired error rate with it.

The baseline has an error rate of 52% and the other systems vary from 51.3% to 52.1%. This is a relative difference of between 0.2–1.3.

Again we take the speed and parameters of the baseline system as reference and calculate the number of mixture components in each system. These results can be seen in table 5.3.

PDF (Gaussian) Type	Components	Parameters	Error Rate	Evaluation Speed
Diagonal (Baseline)	1428	100%	52%	100%
Full	191	90.3%	51.3%	262.3%
Sparse	1305	91.4% - 114.2%	51/6%	136.9%
PPCA	1431	52.2% - 102.3%	52.1%	170.9%
FA	737	77.4%	51.9%	159%

Table 5.3: *Comparison of different PDFs to the second baseline.*



First we discuss each system in isolation:

The Full Covariance Gaussian PDF

For this system we only need 191 mixture components to have a similar error rate (51.3%) to the baseline with 1428 components. This is a 9.7% reduction in parameters. Compared to the previous test, this reduction in parameters is smaller.

With a smaller parameter reduction, the 162.3% longer evaluation time is expected. This is worse than the previous test.

The Sparse Covariance Gaussian PDF

With this test, we need 1305 mixture components to have a comparable error rate (51.6%) to the baseline system. This ranges from 8.6% less parameters to 14.2% more parameters, depending on the effect of the minimum correlation coefficient. It seems that for this test this method tends to have more parameters rather than less.

This system is 36.9% slower to evaluate in this test. This is also a worse result than the previous test.

The PPCA Covariance Gaussian PDF

For this method we need 1431 mixture components to have close to the same error rate (52.1%) than the baseline system. This ranges from 47.8% less to 2.3% more parameters, depending on the effect of the minimum eigenvalue parameter. It seems this method is more likely to have less parameters than the baseline.

This method is still 70.9% slower to evaluate than the baseline system. This is comparative to the results of this method in the first test.

The FA Covariance Gaussian PDF

This method only needs 737 mixture components to have a comparable error rate (51.9%) to the baseline system. That equates to a 22.6% reduction in parameters. This is a huge improvement in parameter reduction over the previous test.

The evaluation of this method took 59% longer than the baseline system. This is comparable to the slower evaluation time in the previous test, even though the parameter ratio seems better.

Overview

Again the results of this test indicates that the baseline system is the fastest. For this test it is at least 36.9% faster than the other systems. It does need more mixture components than the other systems.

Next is the sparse covariance Gaussian system. This system needs more parameters compared to the previous test and is thus slower too. The larger number of parameters indicate the block size of this method is often larger than with the previous tests.

The following system is the FA covariance Gaussian based system. This system is 59% slower than the baseline, which is roughly the same margin as with the previous test. It does have less parameters than the previous test compared to the baseline.

The PPCA covariance Gaussian method is 70.9% slower to evaluate, which is similar to its performance in the previous test. It also has close to the same difference in parameters than before.

The full covariance Gaussian method is 162.3% slower to evaluate than the baseline. This is a worse performance than the previous test. Although it has les parameters than the baseline, the parameter difference is smaller than the first test. This indicates the full covariance Gaussian suffers with smaller training sets.

As a whole, it seems that the diagonal covariance Gaussian method is again the best method to use. The results further confirms the findings of the first test. The other methods had at best the same difference in performance than with the first test. The sparse- and full covariance systems seem to do worse with a small training set.

5.3.3 Third Baseline: Three Mixture Components with a Full Training Set

For the next set of tests, we take the same training test as with the first baseline system, but we decrease the target number of mixture components per PDF to three. Then, as before we train each of the other systems, varying the components to get close to the baseline in error rate. For this test, the spherical covariance Gaussian did give appropriate results and is thus included.

The error rate of the baseline is 57.3% and the other systems vary from 56.2% to 58.4%. This gives a relative difference of about 1.9% in error rate.

Again we take the speed and parameters of the baseline system as reference and calculate the number of mixture components in each system. These results can be seen in table 5.4.

PDF (Gaussian) Type	Components	Parameters	Error Rate	Evaluation Speed
Diagonal	339	100%	57.3%	100%
Full	69	137.4%	56.7%	300.5%
Spherical	705	108.4%	57%	123.5%
Sparse	206	60.8% - 76%	58.4%	154.3%
PPCA	339	52.1% - 102.1%	56.2%	182.2%
FA	206	91.2%	57%	153.7%

Table 5.4: *Comparison of different PDFs to the third baseline.*

First we discuss each system in isolation:

The Full Covariance Gaussian PDF

Here we need 69 mixture components to get close to the baseline error rate (56.7%). The baseline here has 339 components. Comparing the number of parameters, we need 37.3% more parameters for the full covariance Gaussian system in this case. This is the first test where the full Gaussian system needs more parameters than the diagonal Gaussian system.

The evaluation speed is 200.5% slower than the baseline. This is expected, as even with less parameters, this method was slower than the baseline.

The Spherical Covariance Gaussian PDF

The spherical covariance Gaussian system needs 705 mixture components to get a comparable error rate (57%) to the baseline system. For the each component, the spherical covariance Gaussian stores a mean vector and one covariance parameter. This results in 25 parameters per mixture component. The total number of parameters for this system is therefore 8.3% more than the baseline system.

This system takes 23.5% longer to evaluate than the baseline.

The Sparse Covariance Gaussian PDF

For this test we need 206 mixture components to have an error rate (58.4%) close to the baseline system. That equates in a reduction in parameters between 24% to 39.2% over the baseline.

This system takes 54.3% longer to evaluate. Even with less parameters, this is a larger relative gain in evaluation to the previous two sets of tests.

The PPCA Covariance Gaussian PDF

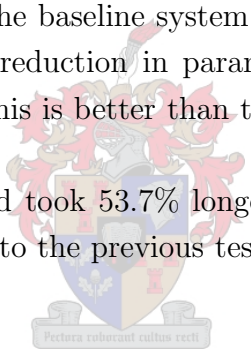
Here we need 339 mixture components to get close to the baseline error rate (56.2%). That gives a number of parameters ranging from 52.1% less than the baseline to 2.1% more than the baseline. This is consistent with the previous tests.

An 82.2% gain in evaluation time over the baseline is seen for this system. This is relatively slower than the previous tests.

The FA Covariance Gaussian PDF

To get close to the error rate of the baseline system (57%), this method needs 206 mixture components. This gives a 8.8% reduction in parameters against the baseline system. If compared to the previous tests, this is better than the first set of tests, but worse than the second set of tests.

During evaluation this method took 53.7% longer. This is an improvement in relative evaluation speed when compared to the previous tests.



Overview

We again see the best performer of all the systems is the diagonal covariance Gaussian system. Although it needs more mixture components than most and sometimes therefor more parameters, it stays the fastest to evaluate.

Next is the spherical covariance Gaussian system. It needs more parameters than the baseline and is 23.5% slower to evaluate. This system even beats the newly implemented methods on evaluation time. Unfortunately the use for this method is limited, illustrated by the fact that no equivalent could be found for the previous two test cases.

The FA covariance Gaussian system is next. It uses less parameters than the baseline, but is 53.7% slower to evaluate. This is a better result than the previous two tests.

Next is the sparse covariance Gaussian. This method has less parameters than the baseline, but is still 54.3% slower to evaluate. This is very close to the performance of the FA covariance system in this test, but with less parameters.

The PPCA covariance Gaussian is 82.2% slower than the baseline to evaluate. It has, at its maximum, about the same parameters as the baseline system.

The full covariance Gaussian is 200.5% slower than the baseline system to evaluate. This test delivered the worst results of all. In this test, the full covariance Gaussian has more

parameters than the baseline system.

It is clear that again the diagonal covariance Gaussian is the best system to use for this case. It is faster by at least 23.5% although it does not have the least number of parameters. This is also the first time we can evaluate the spherical covariance Gaussian. The fact that it does better than the full covariance Gaussian and all the newly implemented methods indicate the price of evaluating off diagonal values of the covariance matrix is computationally much higher than raising the number of mixture components of any method that only regards the covariance along the axes.

5.4 Summary

In this chapter we did some testing on the test systems described in chapter 4. We first did some tests to show the differences of each method. Next we did some tests to evaluate the accuracy and evaluation speed of the PDF systems used. This included three sets of tests which included testing on limited training data and on systems with a limited number of mixture components.

In table 5.5 we show the comparison of all the evaluation speed test results. Each method is shown with its evaluation time as a percentage of the baseline system's time. These results

PDF (Gaussian) type	Evaluation Speed		
	First	Second	Third
Diagonal (Baseline)	100%	100%	100%
Full	242.9%	262.3%	300.5%
Spherical	-	-	123.5%
Sparse	128.9%	136.9%	154.3%
PPCA	168.7%	170.9%	182.2%
FA	161.9%	159%	153.7%

Table 5.5: *Summary of all evaluation speed tests.*

clearly show that in each case the diagonal covariance Gaussian system is the fastest and thus the best choice. This is not surprising, as this method is the tried and tested choice for ASR systems.

The full covariance Gaussian system was the slowest to evaluate in all the cases. Even where the parameters were less than the baseline, this system constantly took more than twice the time to evaluate. This shows that the full covariance Gaussian is an impractical choice for ASR systems.

The spherical covariance Gaussian system could only be tested in one case. Although it outperformed all the newly implemented methods in this test, the result also highlights the shortcoming of this method. For a typical ASR system we want to get the best results. This

method needs so many mixture components to give good error rates that it runs into the danger of running out of data.

In most cases the sparse covariance Gaussian outperformed the PPCA- and FA covariance Gaussian systems in evaluation speed. Only in the last test did the FA covariance Gaussian system had approximately the same performance.

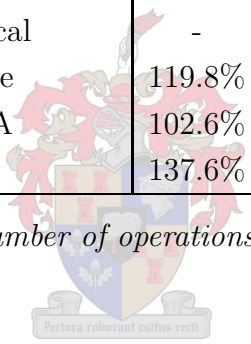
The PPCA covariance Gaussian system gave the worst results for the newly implemented methods. It was consistently slower than the sparse- and FA covariance Gaussian systems.

The FA covariance Gaussian system gave consistent results throughout the tests. It even had better performance on the lower mixture component test.

We also calculate how many (multiplication) operations are needed to score a single vector against each method. This is multiplied by the number of mixture components for each test and the comparison is shown in table 5.6.

PDF (Gaussian) type	Number of Operations		
	First	Second	Third
Diagonal (Baseline)	100%	100%	100%
Full	277%	321%	488.5%
Spherical	-	-	108.3%
Sparse	119.8%	137.1%	91.2%
PPCA	102.6%	106.5%	106.3%
FA	137.6%	105.4%	124.1%

Table 5.6: *Summary of number of operations needed for the tests in table 5.5.*



We see a general tendency in table 5.6 that the sparse-, PPCA and FA covariance systems are slower than the number of operations may suggest. The diagonal covariance Gaussian still has less operations in most cases, making it the preferred method. The rise in evaluation times in this comparison could be attributed to the fact that more data structures and data manipulation is needed to process the evaluation on these methods.

We must also take into account that use of the Tree-Based Adaptive Gaussian Mixture Models (T-BAGMM) do influence the calculation speed for GMMs and they are optimised for a high number of mixture components. Various other methods are used in the PatrecII system to optimise calculation with matrices and other common mathematical issues.

The verdict still remains that the diagonal covariance Gaussian is the best method to use in most (if not all) cases investigated in this thesis. These results not only show us that the diagonal covariance Gaussian is still the dominant performer in ASR systems, but that any attempt to calculate correlation values (full or limited) adds more complexity to the models. Not only did the diagonal covariance Gaussian system constantly outperform the other systems, the one test involving the spherical covariance Gaussian system showed this method is also better than the systems incorporating correlations between dimensions.

This implies that it is more advantageous to increase the mixture components (if possible) of the diagonal covariance Gaussian systems rather than trying to model more covariance information.

Chapter 6

Conclusion

6.1 Concluding Perspective

As we noted earlier in this thesis, the majority of ASR systems use the diagonal covariance Gaussian PDF as base for the higher level PDF structures (such as GMMs and HMMs). The reason for this is its simplicity and its ability to build good practical models of data using a moderate number of mixture components. Practical results have shown the diagonal covariance Gaussian to be a better choice than the full covariance Gaussian for the purposes of ASR [23].

This has for long been the only choice, as the spherical covariance Gaussian lacks the accuracy needed to build a robust data model. Also, the difference in complexity between the diagonal covariance Gaussian and the full covariance Gaussian is substantial. Finding a PDF model (or models in this case) that can fit between these two PDFs in terms of speed and accuracy might be beneficial to building faster and/or more robust statistical data models.

This is where the sparse covariance, PPCA covariance and FA covariance Gaussians come in. All three methods use different methodologies to model more information than the diagonal case. All three of them have been developed to not only model more information, but also to be scalable in terms of how much information should be extracted from the data.

The following issues were examined in this thesis:

- *How do the newly implemented methods compare to the classic methods?* Here we see that each method models more data than the diagonal covariance Gaussian method with their own criteria.

  The sparse covariance Gaussian method preserves blocks consisting of the highest correlated dimensions. The number of dimensions per block and the minimum valid correlation value is predetermined. This leads to longer evaluation times per mixture component and increased accuracy.

  The PPCA covariance Gaussian preserves the information in the principal axes (the number of which is predetermined). The remaining axes are generalised as a spherical

covariance Gaussian noise element. This also requires longer evaluation times over the diagonal covariance Gaussian systems. The accuracy seems very close to that of the diagonal covariance Gaussian system with one principal component. Choosing more should make it more accurate at a further loss of evaluation speed.

The FA covariance Gaussian preserves the information in the factors from factor analysis. The number of factors to retain is predetermined. The remaining information is generalised as a diagonal covariance Gaussian noise element. This method takes longer to evaluate than the diagonal covariance Gaussian systems, but is more accurate.

This shows that the above methods do fit between the diagonal and full covariance Gaussian models. All these methods are also scalable between these two methods via initial parameters.

- *How do the newly implemented methods compare in terms of speed to the classic methods?* We directly compare systems of each type with roughly the same accuracy by measuring the evaluation times of each.

  The sparse covariance Gaussian fared the best of the three implemented methods. It needs less parameters as the diagonal covariance Gaussian system, but still takes longer to evaluate.

  The PPCA covariance Gaussian system has at worse the same number of parameters as the diagonal covariance Gaussian system. It does take somewhat longer to evaluate, only the full covariance Gaussian system is slower.

  The FA covariance Gaussian system needs about the same number of parameters as the diagonal covariance Gaussian system but also takes longer to evaluate, it is however faster than the PPCA covariance Gaussian system.

  These results imply that using the correlation information in ASR is very expensive. Even when the systems modelling correlations have less parameters than the diagonal system, they still take longer to evaluate.

- *How do the newly implemented models compare to the classic methods on limited training data?* We repeat the previous test, but with a severely pruned training set. The reason is to see if any of these methods are particularly useful in systems that have a limited amount of training data.

  The sparse covariance Gaussian system was again the best of the implemented methods. For the worst case, this method has more parameters than the equivalent scoring diagonal covariance Gaussian system. It is again slower too.

  The PPCA covariance Gaussian system again shows the worst results of the implemented methods. It contains, at worse, the same number of parameters than the diagonal covariance Gaussian system, but takes longer to evaluate.

The FA covariance Gaussian system has less parameters than the diagonal covariance Gaussian system, but takes longer to evaluate. It is faster than the PPCA covariance Gaussian system.

It seems that while there are some differences in the number of parameters and evaluation times, the results indicate that the diagonal covariance Gaussian is still the best PDF to use, even on a system with a limited training set. Although the FA covariance Gaussian method did show some improvement, it was not enough to better the faster methods.

- *How do the newly implemented models compare to the classic methods with a limited number of mixture components?* Again we repeat the test process, this time with a full training set, but making the number of mixture components in the system smaller. The aim is to see if any of these methods excel with a restriction on components.

  The sparse covariance Gaussian system gave similar performance than with the previous tests. With less parameters than the diagonal covariance Gaussian system, it was still slower.

  The PPCA covariance Gaussian system again gave the worst results of the implemented methods. Having, at worse, roughly the same number of parameters than the diagonal covariance Gaussian system, it only outperformed the full covariance Gaussian system.

  The FA covariance Gaussian system took roughly the same time to evaluate than the sparse covariance Gaussian system. It uses less parameters than the diagonal covariance Gaussian system, but more than the sparse covariance. Although this is an improvement on the previous tests, the diagonal covariance Gaussian system is still the best.

  This again shows there are not significant enough improvements in these special cases to improve over the efficiency of the diagonal covariance Gaussian PDF. It must be noted that with this test, the spherical covariance Gaussian was also evaluated and also outperformed the implemented methods. This confirms the expense needed to include the off-diagonal values of the covariance matrix.

6.2 Topics for Further Study

In this work we showed the application of some new PDF models to ASR. The testing done in this work was aimed at comparison to the known methods. Some aspects of these methods can be further investigated:

- In testing, we used Linear Discriminant Analysis (LDA) during the post processing step. The same testing (and more) can be done using the Heteroscedastic Linear Discriminant Analysis (HLDA) transform instead. This transform not only separates classes in terms of position, but also in terms of class covariance. Observing the effect

this would have, especially on the PPCA- and FA covariance methods, should yield interesting results.

- Our test system used phoneme recognition. Using these distributions on a word based recogniser using dictionaries may also give interesting results. These systems usually give lower error rates and the performance of these methods could be investigated.

- The testing in this thesis was based on comparative results. Only the number of mixture components were changed during different test cases. With the classic methods the number of components is the only variable to adjust the complexity of the models. With the newly implemented models in this thesis, an element of scalability is added. Further testing of the performance of differently scaled versions of these methods can be done.

- Different signal processing and pattern recognition techniques (such as Support Vector Machines (SVMs)) can be employed on the data and the effect on the results of these PDF models investigated.

- These methods could be employed in other statistical problems such as speaker verification and image processing.

Bibliography

[1] BARBER, D., "Probabilistic Modelling and Reasoning: Factor Analysis and PPCA." http://anc.ed.ac.uk/∼dbarber/pmr/pmr_2003_cont_mixture.pdf. September 2006.

[2] BOYLE, R., "Hidden Markov Models." http://www.comp.leeds.ac.uk/roger/HiddenMarkovModels/html_dev/main.html. October 2006.

[3] CHILDERS, D. G., SKINNER, D. P., and KEMERAIT, R. C., "The Cepstrum: A Guide to Processing." *Proceedings of the IEEE*, October 1977, Vol. 65, No. 10, pp. 1428–1443.

[4] CILLIERS, F., "Tree-based Gaussian Mixture Models for Speaker Verification." Master's thesis, Stellenbosch University, 2005.

[5] DAVIS, S. B. and MERMELSTEIN, P., "Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, August 1980, Vol. 28, No. 4, pp. 357–366.

[6] DELLER, J. R., PROAKIS, J. G., and HANSEN, J. H. L., *Discrete-time Processing of Speech Signals*. New York: Macmillan, 1993.

[7] DEVIJVER, P. A. and KITTLER, J., *Pattern Recognition: A Statistical Approach*. London: Prentice-Hall, 1982.

[8] DU PREEZ, J. A., "Source code for the sparse covariance implemtation in PatrecII.".

[9] DU PREEZ, J. A., *Efficient High-Order Hidden Markov Modelling*. PhD thesis, Stellenbosch University, 1997.

[10] DU PREEZ, J. A., "Thesis-related discussions." 2006.

[11] FUKUNAGA, K., *Introduction to Statistical Pattern Recognition*. Second edition. Academic Press, 1990.

[12] GALES, M. J. F., "Semi-tied covariance matrices for hidden Markov Models." *IEEE Transaction Speech and Audio Processing*, 1999, Vol. 7, pp. 272–281.

[13] GAROFOLO, J. S., LAMEL, L. F., FISHER, W. M., FISCUS, G. J., PALLETT, D. S., and DAHLGREN, N. L., "The DARPA TIMIT acoustic-phonetic continuous speech corpus CDROM." Printed documentation, 1992.

[14] GHAHRAMANI, Z. and HINTON, G. E., "The EM Algorithm for Mixtures of Factor Analyzers." Tech. Rep. CRG-TR-96-1, May 1996.

[15] GOLUB, G. H. and VAN LOAN, C. F., *Matrix Computations*. Baltimore: Johns Hopkins, 1983.

[16] HUANG, X., ACERO, A., and HON, H.-W., *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Prentice-Hall, 2001.

[17] JANKOWSKI, C., "The NTIMIT speech database.." Documentation on NTIMIT CD-ROM, 1992.

[18] KUMAR, N., *Investigation of Silicon-Auditory Models and Generalization of Linear Discriminant Analysis for Improved Speech Recognition*. PhD thesis, John Hopkins University, 1997.

[19] LAY, D. C., *Linear Algebra and its Applications*. Second edition. Addison Wesley Longman, 2000.

[20] MOON, T. K., "The Expectation-Maximization Algorithm." *IEEE Signal Processing Magazine*, November 1996, pp. 47–60.

[21] NIESLER, T., "SS823 course notes." 2001.

[22] PEEBLES JR, P. Z., *Probability, Random Variables and Random Signal Principles*. Fourth edition. McGraw-Hill, 2001.

[23] RABINER, L. R., "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." *Proceedings of the IEEE*, February 1989, Vol. 77, No. 2, pp. 257–286.

[24] RABINER, L. R. and JUANG, B. H., "An Introduction to Hidden Markov Models." *IEEE ASSP Magazine*, January 1986, pp. 4–16.

[25] RABINER, L. R. and JUANG, B. H., *Fundamentals Of Speech Recognition*. New Jersey: Prentice-Hall, 1993.

[26] ROUX, J. C., "Results of the African Speech Technology (AST) Project." `http://www.ast.sun.ac.za/publications/AST_Final.PDF`. October 2006.

[27] SCHWARDT, L. and DU PREEZ, J. A., "PR813 course notes." 2003.

[28] TIPPING, M. E. and BISHOP, C. M., "Mixtures of Probabilistic Principal Component Analyzers." *Neural Computation*, 1999, Vol. 11, pp. 443–482.

[29] ZILL, D. G. and CULLEN, M. R., *Advanced Engineering Mathematics*. Second edition. Sudbury: Jones and Bartlett Publishers, 2000.

Appendix A

Speech Corpora

A.1 The NTIMIT Speech Corpus

The TIMIT speech corpus is a collection of utterances with transcriptions collected by Texas Instruments (TI) and transcribed by the Massachusetts Institute of Technology (MIT). The NTIMIT corpus is a telephone bandwidth version of the TIMIT corpus and is in all other respects the same. The corpus contains 6300 sentences; 10 sentences spoken by each of the 630 speakers. The speakers are from eight different dialects found in the United States. The transcriptions contains time indicators for all the speech in each utterance. Detailed information on these corpora can be found in [13, 17].
The TIMIT speech was sampled at 16kHz and stored using 16 bits/sample before it was put through the telephone channels. The NTIMIT speech corpus is the result of putting the TIMIT data through actual telephone channels and was not not simulated. These channels were varied to get samples of various conditions and the utterances were time-aligned to the TIMIT corpus to make use of the original transcriptions.
The NTIMIT corpus consists of three sets for each speaker: The *sa*, *sx* and *si* sets. The *sa* set are the same sentences for all the speakers, the *sx* set are read from a list of 450 phonetically balanced sentences and the *si* set are read from a list of 1890 phonetically diverse sentences.

A.2 The African Speech Technology Speech Corpus

The African Speech Technology (AST) speech corpus consists of five different South African languages. These are Afrikaans, English, isiXhosa, isiZulu and Sesotho. Each language has a mixture of read or spontaneous mother-tongue speakers. There is roughly 38 to 40 utterances per speaker. The speech was recorded over standard telephone and cellphone conditions. The utterances are transcribed, but only the onset and the termination times are recorded. The time of occurrence of non-speech events are also indicated. The speech itself has no time indicators.
For this thesis we only considered the English language. This set is again divided in

dialects. These are Afrikaans English, Asian English, Black English, Coloured English and Standard English. For this thesis we used all of the above, except the Black English. More information can be found in [26].
The corpus was recorded over Integrated Services Digital Network (ISDN) lines from cellphones and land line telephones. The utterances were sampled at 8kHz and stored using 8 bit/sample over a single channel. There are approximately between $6\frac{1}{2}$ - 8 hours of speech in each dataset with about 250 - 350 utterances per set.
This corpus also has a separate test dataset which consists of roughly 10% of the total data. This separate test set was used exclusively for testing purposes.

Appendix B

Code Implementations

Following is the code implementation in Matlab of all the PDF models discussed in this chapter. The code is given for the calculation of each PDF type, followed by a generic procedure for the calculation of a GMM for one of these PDF types:

```
function [mu, Sigma] = fullGauss(x)

% function [mu, Sigma] = fullGauss(x)
% reads a set of training vectors and outputs the mean and
% full covariance matrix

% x - training vectors

% mu - mean vector for training set
% Sigma - Full covariance matrix for training set


mu = mean(x')';
Sigma = cov(x');

------------------------------------------------------------

function p = fullscore(mu, Sigma, y)

% function p = fullscore(mu, Sigma, y)
% reads in statistics for a full Gaussian
% plus a test vector set and returns the
% probability of the set being generated by
% the full Gaussian. Returns probability in
% log likelihood form.
```

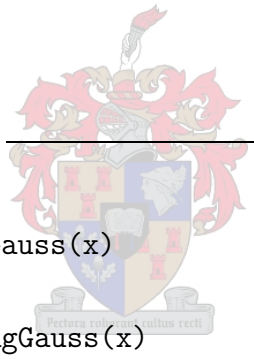
```
% mu - mean of full Gaussian
% Sigma - covariance matrix of full Gaussian
% y - set of test vectors for scoring

% p - probability that all the vectors of y
%     was generated by this model in log
%     likelihood form.

[dim N] = size(y);
prob = [];
Sigmainv = inv(Sigma);
SigmadetTerm = (dim/2)*log(2*pi) + 0.5*log(det(Sigma));

for i = 1:N
 prob = [prob, (-0.5*(y(:,i) - mu)'*Sigmainv*(y(:,i) - mu)
                                          - SigmadetTerm)];
end

p = sum(prob);
```



```
function [mu, Sigma] = diagGauss(x)

% function [mu, Sigma] = diagGauss(x)
% reads a set of training vectors and outputs the mean and
% diagonal covariance matrix (as a vector)

% x     - training vectors

% mu    - mean vector for training set
% Sigma - diagonal covariance matrix for training set
%         (as a vector)

[dim N] = size(x);

mu = mean(x')';
Sigma  = sum(((x - repmat(mu,1,N)).^2)')'/(N-1);

-------------------------------------------------------
```

```
function p = diagscore(mu, Sigma, y)

% function p = diagscore(mu, Sigma, y)
% reads in statistics for a diagonal Gaussian
% plus a test vector set and returns the
% probability of the set being generated by
% the diagonal Gaussian. Returns probability
% in log likelihood form.

% mu - mean of diagonal Gaussian
% Sigma - covariance matrix of diagonal Gaussian
% y - set of test vectors for scoring

% p - probability that all the vectors of y
%     was generated by this model in log
%     likelihood form.

[dim N] = size(y);
prob = [];
SigmadetTerm = (dim/2)*log(2*pi) + 0.5*sum(log(Sigma));

for i = 1:N
 prob = [prob, ((-0.5*sum(((y(:,i) - mu).^2)./Sigma))
                                    -SigmadetTerm)];
end

p = sum(prob);
```

```
function [mu, Sigma] = spherGauss(x)

% function [mu, Sigma] = spherGauss(x)
% reads a set of training vectors and outputs the mean and
% spherical covariance matrix (as a single value)

% x     - training vectors

% mu    - mean vector for training set
% Sigma - spherical covariance matrix for training set
%         (as a single value)
```

```
[dim N] = size(x);

mu = mean(x')';
Sigma = sum(sum(((x-repmat(mu,1,N)).^2)))/(dim*(N-1));

-------------------------------------------------------------

function p = spherscore(mu, Sigma, y)

% function p = spherscore(mu, Sigma, y)
% reads in statistics for a spherical Gaussian
% plus a test vector set and returns the
% probability of the set being generated by
% the spherical Gaussian. Returns probability
% in log likelihood form.

% mu - mean of spherical Gaussian
% Sigma - covariance of spherical Gaussian
% y - set of test vectors for scoring

% p - probability that all the vectors of y
% was generated by this model in log
% likelihood form.

[dim N] = size(y);
prob = [];
SigmadetTerm = (dim/2)*log(2*pi*Sigma);

for i = 1:N
 prob = [prob, (-((y(:,i)-mu)'*(y(:,i)-mu))/(2*Sigma)
                                        - SigmadetTerm)];
end

p = sum(prob);
```

```
function [mu, Sigma] = sparseGauss(x,maxBlocks,minCorr)

% function [mu, Sigma] = sparseGauss(x,maxBlocks,minCorr)
% reads a set of training vectors, a maximum block size value,
```

```
% a minimum correlation coefficient value and outputs the mean
% and a structure containing all the block indices and the
% corresponding covariance values

% x - training vectors
% maxBlocks - the maximum block size allowed
% minCorr - the minimum correlation coefficient value to
%           evaluate

% mu - mean vector for training set
% Sigma - structure containing the diagonal (diag), the
%         blocks (Block) and off-diagonal (OffDiag) values
%         of the sparse covariance matrix. Block contains
%         vectors with the indexes and OffDiag contains vectors
%         with the matrix entries for the blocks

[dim N] = size(x);


mu = mean(x')';

C = cov(x');

Sigma = makeSparse(C, maxBlocks, minCorr);

------------------------------------------------------------------

function Sigma = makeSparse(C, maxBlocks, minCorr)

dim = length(C);
Sigma.diag = diag(C);

% Calculate correlation coefficient matrix

for r = 2:dim
 for c = 1:(r-1)
  Corr(r,c) = C(r,c)/sqrt(C(r,r) * C(c,c));
  Corr(c,r) = Corr(r,c);
 end
end
```

```
% Remove values below minCorr and find the maximum
% correlation value

maxV = 0;
maxR = -1;
maxC = -1;
for r = 1:dim
 Corr(r,r) = 0;
 for c = 1:(r-1)
  Corr(r,c) = abs(Corr(r,c));
  if (Corr(r,c) < minCorr)
   Corr(r,c) = 0;
   Corr(c,r) = 0;
  else
   Corr(c,r) = Corr(r,c);
  end
  if (Corr(r,c) > maxV)
   maxV = Corr(r,c);
   maxR = r;
   maxC = c;
  end
 end
end

% Process the correlation values from the largest
% and build the blocks until all correlations above
% the minimum have been evaluated

blocks = 0;
while (maxV > minCorr)
 rBlocks = -1;
 cBlocks = -1;
 for b = 1:blocks
  if (sum(Block(b).vec == maxR) == 1)
   rBlocks = b;
  end
  if (sum(Block(b).vec == maxC) == 1)
   cBlocks = b;
  end
 end
```

```
tempBlock = sort([maxR maxC]);
if (rBlocks > 0)
 for i = 1:length(Block(rBlocks).vec)
  if (sum(tempBlock == Block(rBlocks).vec(i)) == 0)
   tempBlock = sort([tempBlock Block(rBlocks).vec(i)]);
  end
 end
end
if (cBlocks > 0)
 for i = 1:length(Block(cBlocks).vec)
  if (sum(tempBlock == Block(cBlocks).vec(i)) == 0)
   tempBlock = sort([tempBlock Block(cBlocks).vec(i)]);
  end
 end
end
if (length(tempBlock) <= maxBlocks)
 if ((rBlocks > 0) & (cBlocks < 0))
  Block(rBlocks).vec = tempBlock;
 end
 if ((rBlocks < 0) & (cBlocks > 0))
  Block(cBlocks).vec = tempBlock;
 end
 if ((rBlocks < 0) & (cBlocks < 0))
  blocks = blocks + 1;
  Block(blocks).vec = tempBlock;
 end
 if ((rBlocks > 0) & (cBlocks > 0))
  if (rBlocks == cBlocks)
   Block(rBlocks).vec = tempBlock;
  else
   Block(min([rBlocks cBlocks])).vec = tempBlock;
   tempCount = max([rBlocks cBlocks]);
   while (tempCount < blocks)
    Block(tempCount).vec = Block(tempCount+1).vec;
    tempCount = tempCount + 1;
   end
   blocks = blocks - 1;
  end
 end
end
```

```
 Corr(maxR, maxC) = 0;
 Corr(maxC, maxR) = 0;
 maxV = 0;
 maxR = -1;
 maxC = -1;
 for r = 1:dim
  for c = 1:(r-1)
   if (Corr(r,c) > maxV)
    maxV = Corr(r,c);
    maxR = r;
    maxC = c;
   end
  end
 end
end

% Write the off-diagonal covariance entries needed
% by the blocks in corresponding vectors

for b = 1:blocks
 Sigma.Block(b) = Block(b);
 setSz = length(Block(b).vec);
 Sigma.OffDiag(b).vec = [];
 for j = 2:setSz
  cCnt = Block(b).vec(j);
  for k = 1:(j-1)
   rCnt = Block(b).vec(k);
   Sigma.OffDiag(b).vec = [Sigma.OffDiag(b).vec C(rCnt, cCnt)];
  end
 end
end

-----------------------------------------------------------------

function p = sparsescore(mu, Sigma, y)

% function p = sparsescore(mu, Sigma, y)
% reads in statistics for a sparse Gaussian
% plus a test vector set and returns the
% probability of the set being generated by
```

```
% the sparse Gaussian. Returns probability
% in log likelihood form.

% mu - mean of sparse Gaussian
% Sigma - structure containing covariance of
%         sparse Gaussian
% y - set of test vectors for scoring

% p - probability that all the vectors of y
%     was generated by this model in log
%     likelihood form.

[dim N] = size(y);
ym = y - repmat(mu,1,N);


% invert sparse Gaussian and save in
% same structure
for i = 1:dim
 Mat(i,i) = Sigma.diag(i);
end
if (length(struct2cell(Sigma)) == 3)
 blocks = length(Sigma.Block);
 for b = 1:blocks
  valCnt = 1;
  for j = 2:length(Sigma.Block(b).vec);
   cCnt = Sigma.Block(b).vec(j);
   for k = 1:(j-1)
    rCnt = Sigma.Block(b).vec(k);
    Mat(cCnt,rCnt) = Sigma.OffDiag(b).vec(valCnt);
    Mat(rCnt,cCnt) = Sigma.OffDiag(b).vec(valCnt);
    valCnt = valCnt + 1;
   end
  end
 end
end
invMat = inv(Mat);
diagInv = diag(invMat);
if (length(struct2cell(Sigma)) == 3)
 for b = 1:blocks
```

```
  setSz = length(Sigma.Block(b).vec);
  OffDiagInv(b).vec = [];
  for j = 2:setSz
   cCnt = Sigma.Block(b).vec(j);
   for k = 1:(j-1)
    rCnt = Sigma.Block(b).vec(k);
    OffDiagInv(b).vec = [OffDiagInv(b).vec invMat(rCnt, cCnt)];
   end
  end
 end
end
detMat = det(Mat);
prob = [];
SigmadetTerm = (dim/2)*log(2*pi) + 0.5*log(detMat);

% Calculate log likelihood
for i = 1:N
 quad = 0;
 if (length(struct2cell(Sigma)) == 3)
  for b = 1:blocks
   valCnt = 1;
   setSz = length(Sigma.Block(b).vec);
   for j = 2:setSz
    cCnt = Sigma.Block(b).vec(j);
    for k = 1:(j-1)
     rCnt = Sigma.Block(b).vec(k);
     quad = [quad ym(rCnt,i)*ym(cCnt,i)*OffDiagInv(b).vec(valCnt)];
     valCnt = valCnt + 1;
    end
   end
  end
 end
 quad = sum(quad);
 quad = quad*2;
 quad = quad + sum(((ym(:,i).^2).*diagInv));
 prob = [prob (-0.5*quad - SigmadetTerm)];

end
p = sum(prob);
```

```
function [mu, Sigma] = PPCAGauss(x,dimRedux,minInput)

% function [mu, Sigma] = PPCAGauss(x,dimRedux,minInput)
% reads a set of training vectors, a number of dimensions
% to explicitly model and the minimum eigenvalue to consider
% and outputs the mean and a structure containing the modelled
% information and noise parameter

% x - training vectors
% dimRedux - number of dimensions to consider
% minInput - minimum value of eigenvalue to consider

% mu - mean vector for training set
% Sigma - structure containing the modelled information (W) and
%         the noise parameter (sigma2)

[dim N] = size(x);
mu = mean(x')';

C = cov(x');

[U, D, V] = svd(C); % get eigenvalues and vectors in descending order

newDimRedux = dimRedux;
for i = 1:dimRedux
 if (D((dimRedux + 1 - i),(dimRedux + 1 - i)) < sum(diag(D))*minInput)
  newDimRedux = newDimRedux - 1;
 end
end
dimRedux = newDimRedux;

Sigma.sigma2 = (sum(diag(D(dimRedux+1:dim, dimRedux+1:dim))))
                                                    /(dim-dimRedux);

Sigma.W = U(:,1:dimRedux)*(D(1:dimRedux,1:dimRedux)
                           - diag(ones(dimRedux,1))*Sigma.sigma2)^0.5;

-----------------------------------------------------------------------

function p = PPCAscore(mu, Sigma, y)
```

```
% function p = PPCAscore(mu, Sigma, y)
% reads in statistics for a PPCA
% Gaussian plus a test vector set and returns the
% probability of the set being generated by the
% PPCA Gaussian. Returns probability in log
% likelihood form.

% mu - mean of PPCA Gaussian
% Sigma - structure containing covariance of
%         PPCA Gaussian
% y - set of test vectors for scoring

% p - probability that all the vectors of
%     y was generated by this model in log
%     likelihood form.

[dim N] = size(y);
[D M] = size(Sigma.W);
ym = y - repmat(mu,1,N);
prob = [];

Sigmainv = inv(Sigma.sigma2*diag(ones(M,1)) + Sigma.W'*Sigma.W);
SigmadetTerm = (dim/2)*log(2*pi) + 0.5*log(det(Sigma.W*Sigma.W'
                                + diag(ones(D,1))*Sigma.sigma2));

for i = 1:N
 ya = Sigma.W'*ym(:,i);
 prob = [prob (-0.5*((ym(:,i)'*ym(:,i)-ya'*Sigmainv*ya)/Sigma.sigma2)
                                                        - SigmadetTerm)];
end

p = sum(prob);
```

```
function [mu, Sigma] = FAGauss(x,dimRedux,W,Psi)

% function [mu, Sigma] = FAGauss(x,dimRedux,W,Psi)
% reads a set of training vectors, a number of dimensions
% to explicitly model and the previous iteration values of
```

```
% W and Psi, calculates one iteration and returns the mean
% and a structure containing the modelled information and
% noise vector

% x - training vectors
% dimRedux - number of dimensions to consider
% W - previous iteration (or initial) W
% Psi - previous iteration (or initial) Psi

% mu - mean vector for training set
% Sigma - structure containing the modelled information (W)
% and noise vector (Psi)

[dim N] = size(x);
mu = mean(x')';

C = cov(x');

Beta = W'*(diag(Psi.^-1) - diag(Psi.^-1)*W*(eye(dimRedux)
                 + W'*diag(Psi.^-1)*W)^-1*W'*diag(Psi.^-1));

E1 = C*Beta'*(N-1);
E2 = N*(eye(dimRedux) - Beta*W) + Beta*C*Beta'*(N-1);

W = E1*E2^-1;
Psi = diag(C*(N-1) - W*E1')/N;

Sigma.W = W*sqrt(N/(N-1));
Sigma.Psi = Psi*N/(N-1);

-------------------------------------------------------------

function p = FAscore(mu, Sigma, y)

% function p = FAscore(mu, Sigma, y)
% reads in statistics for a FA Gaussian
% plus a test vector set and returns the
% probability of the set being generated
% by the FA Gaussian. Returns probability
% in log likelihood form.
```

```
% mu - mean of FA Gaussian
% Sigma - structure containing the covariance
%         of FA Gaussian
% y - set of test vectors for scoring

% p - probability that all the vectors of y
%     was generated by this model in log
%     likelihood form.

[dim N] = size(y);
[D M] = size(Sigma.W);
ym = y - repmat(mu,1,N);
prob = [];
if (M > 0)
 Sigmainv = inv(eye(M) + Sigma.W'*((Sigma.Psi.^-1).*Sigma.W));
 SigmadetTerm = (dim/2)*log(2*pi) + 0.5*log(det(Sigma.W*Sigma.W'
                                          + diag(Sigma.Psi)));
else
 SigmadetTerm = (dim/2)*log(2*pi) + 0.5*log(det(diag(Sigma.Psi)));
end

for i = 1:N
 if (M > 0)
  ya = (repmat(Sigma.Psi.^-1,1,M).*Sigma.W)'*ym(:,i);
  prob = [prob (-0.5*((ym(:,i)'*(Sigma.Psi.^-1.*ym(:,i))
                              - ya'*Sigmainv*ya)) - SigmadetTerm)];
 else
  prob = [prob (-0.5*((ym(:,i)'*(Sigma.Psi.^-1.*ym(:,i))))
                                                  - SigmadetTerm)];
 end
end

p = sum(prob);
```

```
function GmmStruct  = GMM(x,num,itr)

% function GmmStruct = GMM(x,num,itr)
% Trains a GMM with num number of components using
```

```
% the training vectors in x for a maximum of itr iterations

% GmmStruct - a array of structures containing 3 fields:
% Sigma, mu and prior, where Sigma is the structure containing
% the covariance information, mu is the mean value and prior
% is the prior probability of the Gaussian.

% x - the set of training vectors
% num - the number of mixtures to train
% itr - the maximum number of iterations

[dim, N] = size(x);
X(1).vec = x;

% initialise and assign the Gaussian and mean for the first cluster
% Could be diag, full, sphere, PPCA or FA (with initialisation)
% e.g.: [GmmStruct(1).mu GmmStruct(1).Sigma] = diagGauss(x);

GmmStruct(1).prior = 1;

% split the cluster with the largest 'distortion' until the desired
% number of clusters are reached
for i = 2:num
 Dist = [];
 for j = 1:(i-1)
  [dim, subN] = size(X(j).vec);
  distort = [];
  for k = 1:subN
   distort = [distort ((GmmStruct(j).mu - X(j).vec(:,k))'*(GmmStruct(j).mu
                                                        - X(j).vec(:,k)))];
  end
  Dist = [Dist sum(distort)];
 end
 [maxVal, I] = max(Dist);
 Xtemp = X(I).vec;
 [dim, subN] = size(Xtemp);
 [U, D, V] = svd(cov(Xtemp'));
 muTemp(1).vec = GmmStruct(I).mu + V(:,1);
 muTemp(2).vec = GmmStruct(I).mu - V(:,1);
 Xt(1).vec = [];
```

```
 Xt(2).vec = [];
 clear distort;
 for j = 1:subN
  distort(1) = ((muTemp(1).vec - Xtemp(:,j))'*(muTemp(1).vec - Xtemp(:,j)));
  distort(2) = ((muTemp(2).vec - Xtemp(:,j))'*(muTemp(2).vec - Xtemp(:,j)));
  [minVal, J] = min(distort);
  Xt(J).vec = [Xt(J).vec Xtemp(:,j)];
 end
 X(I).vec = Xt(1).vec;
 [dim, subN1] = size(Xt(1).vec);
 X(i).vec = Xt(2).vec;
 [dim, subN2] = size(Xt(2).vec);

 % initialise and assign the Gaussian and mean for the first cluster
 % Could be diag, full, sphere, PPCA or FA (with initialisation)
 % e.g.: [GmmStruct(I).mu, GmmStruct(I).Sigma] = diagGauss(Xt(1).vec);
 %       [GmmStruct(i).mu, GmmStruct(i).Sigma] = diagGauss(Xt(2).vec);

 GmmStruct(I).prior = subN1/N;
 GmmStruct(i).prior = subN2/N;
end

% Do EM-algorithm until convergence or maximum iterations are reached
LLold = 0;
for it = 1:itr
 for i = 1:N
  for j = 1:num
   % calculate the score for the Gaussian
   % e.g.: pk(j) = diagscore(GmmStruct(j).mu, GmmStruct(j).Sigma, x(:,i))
   %                                          + log(GmmStruct(j).prior);
  end
  Xm = max(exp(pk));
  Pfin(i) = log(Xm) + log(sum(exp(pk - log(Xm))));
  pkn(i,:) = pk - Pfin(i);
 end
 maxVec = log(max(exp(pkn)));
 Pk = exp((maxVec + log(sum(exp((pkn - repmat(maxVec,N,1)))))) - log(N));
 means = (x*exp(pkn))/diag(exp(maxVec + log(sum(exp((pkn
                                        - repmat(maxVec,N,1)))))));
```

```
 for i = 1:num
  GmmStruct(i).prior = Pk(i);
  GmmStruct(i).mu = means(:,i);
  xm = x - repmat(means(:,i),1,N);
  fullCov = ((exp(repmat(pkn(:,i),1,dim))'.*xm)*xm')/exp(maxVec(i)
                                  + log(sum(exp(pkn(:,i) - maxVec(i)))));
  % recalculate covariance structure using the full covariance:
  % i.e. diagonal, spherical, PPCA or FA (one iteration)
  % e.g.: GmmStruct(i).Sigma = diag(fullCov);
 end
 LLnew = sum(Pfin);
 if (LLold == 0)
  change = 1;
 else
  change = (LLnew - LLold)/abs(LLold);
 end
 LLold = LLnew;
 disp(['Iteration: ', int2str(it)]);
 disp(['Log likelihood: ', num2str(LLnew)]);
 disp(['improvement: ', num2str(change)]);
 if (change < (5 * 10^-4))
  disp('Converging!')
  break
 end;
end
```