TiNONS Project - Speaker Recognition: Appendix

Kasper Nielsen - 10731
Alexander Rasborg Knudsen - 20107083

June 2, 2014

Contents

1 Dataset

1.1 Description

The data that we use to test individual classification methods in this project consists of recordings from 4 speakers repeatedly saying the numbers 0 through nine. From this set one speaker was removed, because significantly fewer recordings ware made of this. Hence the amount of data points less fewer than was desired.

Out of the dataset, 15 utterings of each digit ware selected randomly from each speaker, classified and put into the training set. Another 5 utterings of each digit from each speaker was then classified ad selected as test set.

The data is sampled at 48 $kHz, 16\frac{bits}{sample}$

The data is provided by [1]

1.2 Features

When doing speaker recognition it is advantageous to classify on the basis of extracted features from speech data, rather than the audio samples themselves [2].

Therefore some features were extracted, using the following techniques before further processing.

1.3 Mel-frequency Cepstral Coefficients

In this project, the features used for classification are based on Mel-frequency Cepstral Coefficients (MFCC). They have proven effective for use in speaker recognition, in other implementations [2].

A mel-frequency cepstrum is dscribed as a cepstrum (a nonlinear spectrum-of-a-spectrum) on a mel-frequency scale. The mel-scale is a non-linear frequency scale that mimics the characteristics of the human ear. It is approximately linear below 1000 Hz and logarithmic above. This makes it ideal for audio and human speech processing.

Besides the bare MFCCs, the 0^{th} order coefficients, which are equivalent to the energy, and the delta and double-delta coefficients, which are the temporal derivatives $\frac{dc}{dt}$ and double-derivatives $\frac{d^2c}{dt}$ respectively, are used as features.

1.4 Feature extraction

1.4.1 Framing

In this project we have chosen a to segment data into short frames, and extract features for classification on the basis of these. We have chosen a frame length of 256 *samples* with a spacing of 100 *samples*.

This means that we have a frame length of

$$t_{framelength} = \frac{N_{frame}}{F_s} = \frac{256}{48\ kHz} = 5.33\ ms \tag{1.1}$$

and a frame interval of

$$t_{frameinterval} = \frac{N_{interval}}{F_s} = \frac{100}{48\ kHz} = 2.08\ ms \tag{1.2}$$

1.4.2 MFCC feature calculation

The extraction of MFCC features from audio samples is done in MATLAB, using an open-source toolbox called Voicebox [3]. It calculates the MFCCs in the following way: (See Figure 1.1)

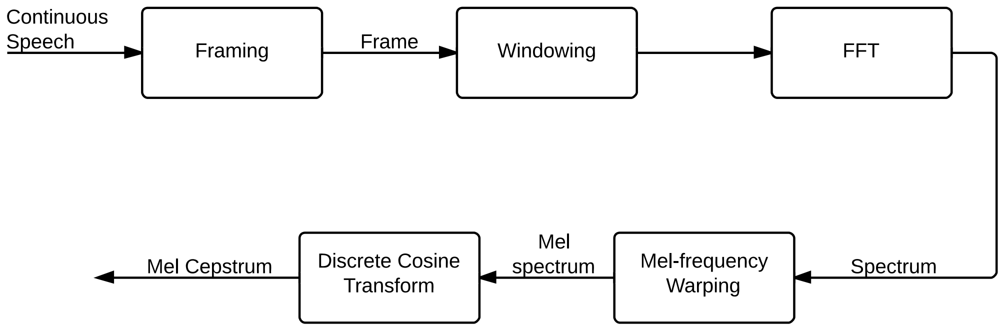

Figure 1.1: Flowchart of MFCC calculation

1. The current frame is windowed with a Hamming window.
2. The frame is Fourier transformed.
3. The power of the spectrum is mapped to a mel-scale, using triangular overlapping windows. (See Figure 1.2)
4. Take the logarithm of the power a each mel-frequency
5. Take the Discrete Cosine Transform (DCT) of the log-powers as is it were a signal.
6. The MFCCs are the amplitudes of the resulting spectrum.

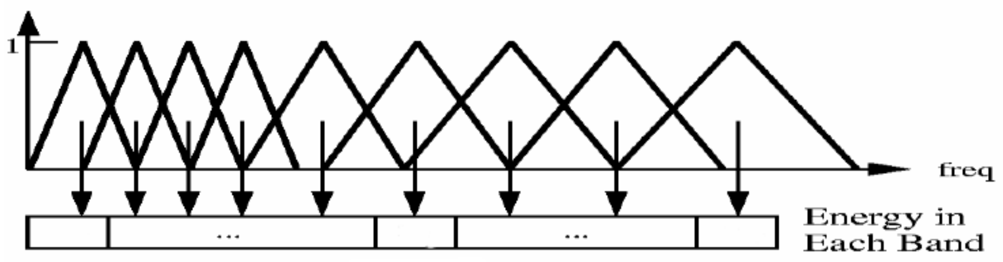

Figure 1.2: an example of a mel-spaced filterbank

2 Evaluation of classification methods

The individual classification methods are evaluated with datasets of varying complexities. Most notably, with speakers uttering the same single digit ("*ZERO*"), two different digits ("*ZERO*" and "*ONE*"), and ten different digits ("*ZERO*", "*ONE*" through "*NINE*").

To evaluate a classifier, the confusion matrix is an ideal way of doing so. The confusion matrix can show the classifications sensitivity, precision and accuracy for each class. The terms used to describe this is: false positive (*FP*), true positive (*TP*), false negative (*FN*) and true negative (*TN*). The true terms are classes that are correctly classified and false terms are incorrectly classified. The sensitivity is the probability of classifying the inputs as class X for a input that are class X.

$$\texttt{sensitivity}(X) = \frac{TP_X}{TP_X + \Sigma FN_X} \tag{2.1}$$

The precision is the probability that the estimate of class X is correct.

$$\texttt{precision}(X) = \frac{TP_X}{TP_X + \Sigma FP_X} \tag{2.2}$$

The accuracy is the probability that the classification of any given class is correct, where N = total number of tests.

$$\texttt{accuray}(X) = \frac{\Sigma TP_X}{\mathrm{N}} \tag{2.3}$$

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	6300.0	0.0	0.0	100.0
Estimate Mose	10.0	5790.0	100.0	98.1
Estimate Simon	600.0	1120.0	6810.0	79.8
Sensitivity [%]	91.2	83.8	98.6	91.2

Table 2.1: Example of confusion matrix. GMM classification with 3 speakers and 2 digits spoken

Above in Table 2.1 an example of a confusion matrix in shown. The far right column shows the precision as calculated with (2.2), the bottom row shows the sensitivity as calculated with (2.1) and the bottom right field shows the overall accuracy as calculated with (2.3). This is the way, along with some plots of target vs. estimate, that the results of using various methods for classification will be presented in this project.

The evaluation of classifiers will primarily be based on overall accuracy of the classifier.

3 Linear classification

3.1 Theory

In linear classification the decision surfaces are linear functions of the input vector $\mathbf{x}$. The decision surfaces are defined by D-1 dimensional hyperplane in the D dimensional input space. the target of the classification is labelled in the target variable $\mathbf{t}$, using the target values to represent class labels. In the case of two-class problems, a single target variable can be represented by $t \in \{0, 1\}$ where $t = 1$ represents class C_1 and $t = 0$ represents class C_2. If the is more than 2 classes ($K > 2$), then $\mathbf{t}$ is a vector of length K. For Class C_j the element t_j takes value 1 and all other elements t_k of $\mathbf{t}$ are zero. In this project there are 3 different speakers, $K = 3$ then the target vector for class 3 be $\mathbf{t} = (0, 0, 1)^T$. The value of t_k can be interpreted as the probability of the given class being class C_k. To assign each vector $\mathbf{x}$ with a specific class, a number of different approaches can used to classify. One way is using a discriminant function, with a linear predictor $y(\mathbf{x}, \mathbf{w})$ given by the parameter $\mathbf{w}$ and the input vector set $\mathbf{x} = (x_1, ..., x_D)^T$, which is linear. The linear discriminant function in its simplest form

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 \tag{3.1}$$

Where w_0 is the bias and $\mathbf{w}$ is the weight vector. The decision boundary corresponds to $y(\mathbf{x}) = \textit{constant}$, and hence $\mathbf{w}^T\mathbf{x} + w_0 = \textit{constant}$ therefore the decision boundary is a linear function.

3.2 Method

The process to apply the linear classifier to the dataset, is described below. The result of the linear classifier is evaluated by applying a confusion matrix and calculating the accuracy. The linear classifier is applied to the training dataset, containing feature vectors ($\mathbf{x}_n$) and the target vectors ($\mathbf{t}_n$). The vectors are on the form:

$$\tilde{\mathbf{X}} = \begin{bmatrix} \mathbf{x}_1^T & 1 \\ \mathbf{x}_2^T & 1 \\ \multicolumn{2}{c}{...} \\ \mathbf{x}_n^T & 1 \end{bmatrix}, \ \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \mathbf{t}_2^T \\ ... \\ \mathbf{t}_n^T \end{bmatrix} \tag{3.2}$$

To determine the weight matrix $\tilde{\mathbf{W}}$:

$$\tilde{\mathbf{W}} = \tilde{\mathbf{X}}^{\dagger}\mathbf{T} \approx (\tilde{\mathbf{X}}^T\tilde{\mathbf{X}} + \mathbf{I})^{-1}\tilde{\mathbf{X}}^T\mathbf{T} \tag{3.3}$$

The are a problem with the entity $\tilde{\mathbf{X}}^T\tilde{\mathbf{X}}$ having small eigenvalues, the solution is to add the identity matrix in the calculations of the pseudo inverse matrix $\tilde{\mathbf{X}}^{\dagger}$. This is done as a regularization, which can be seen as a penalty for large values in $\tilde{\mathbf{W}}$.

When $\tilde{\mathbf{W}}$ is calculated from the training set, it is tested on the test dataset containing feature vectors and their corresponding targets. The result of this equation

$$\mathbf{y}(\mathbf{x}) = \tilde{\mathbf{W}}^T \tilde{x},\ \tilde{x} = \left[\begin{array}{cc} \mathbf{x} & 1 \end{array} \right] \tag{3.4}$$

shows the most likely class for each of the given feature vectors. The resulting vector is placed in a vector $\mathbf{t}_{est}$, which contains the classification and ranges over the three classes $\{1, 2, 3\}$.

3.3 Results

3.3.1 Single digit:

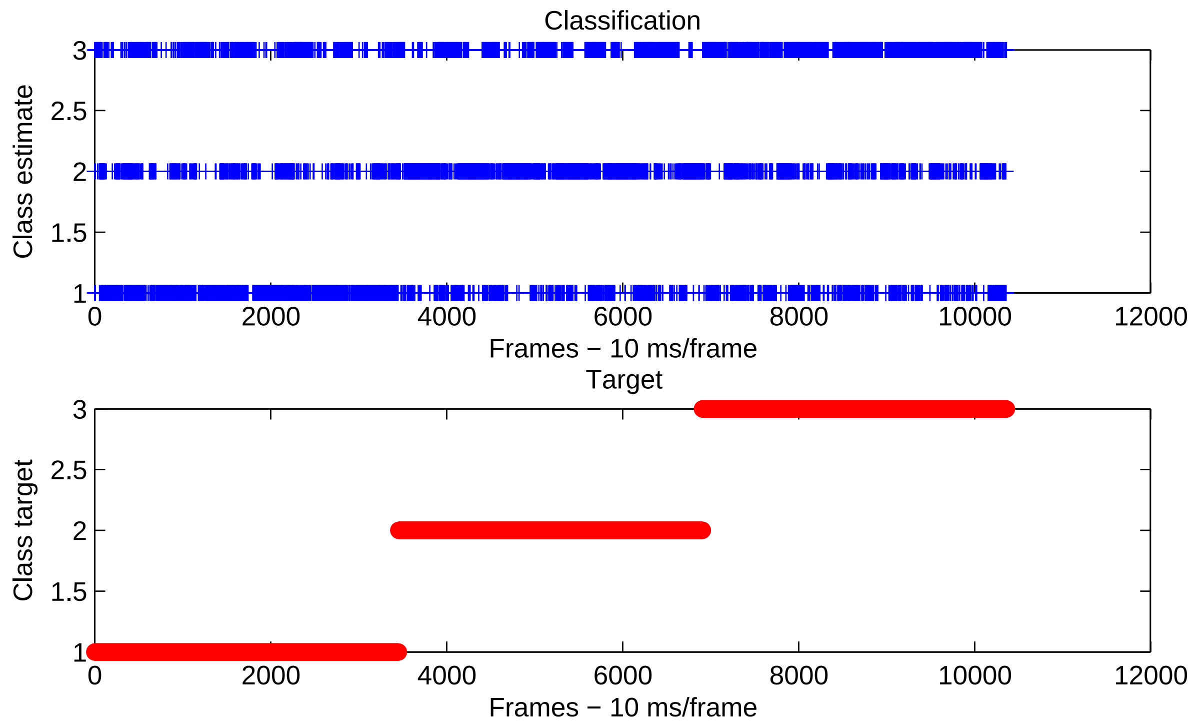

Figure 3.1: Results of using linear classifiers and single digit spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	2041.0	557.0	699.0	61.9
Estimate Mose	562.0	1816.0	820.0	56.8
Estimate Simon	851.0	1081.0	1935.0	50.0
Sensitivity [%]	59.1	52.6	56.0	55.9

Table 3.1: Confusion matrix - 1 digits

3.3.2 2 digits:

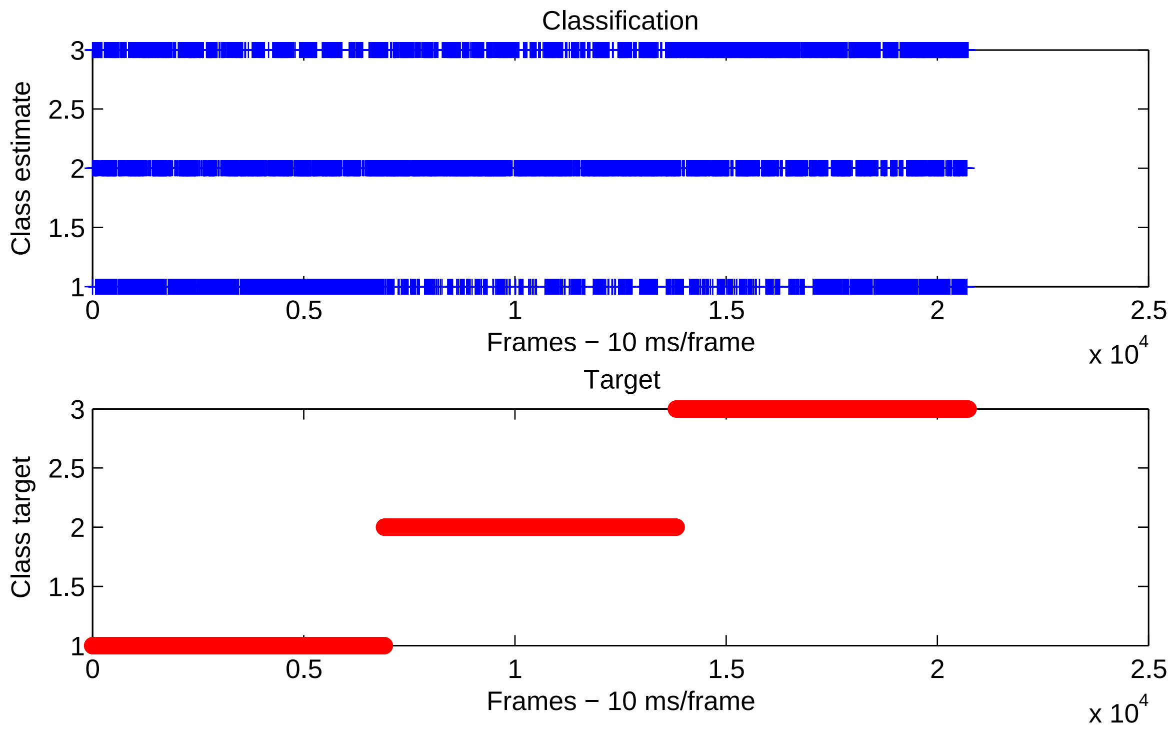

Figure 3.2: Results of using linear classifiers and two digits spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	3169.0	830.0	1535.0	57.3
Estimate Mose	2163.0	3896.0	1818.0	49.5
Estimate Simon	1578.0	2184.0	3557.0	48.6
Sensitivity [%]	45.9	56.4	51.5	51.2

Table 3.2: Confusion matrix - 2 digits

3.3.3 10 digits:

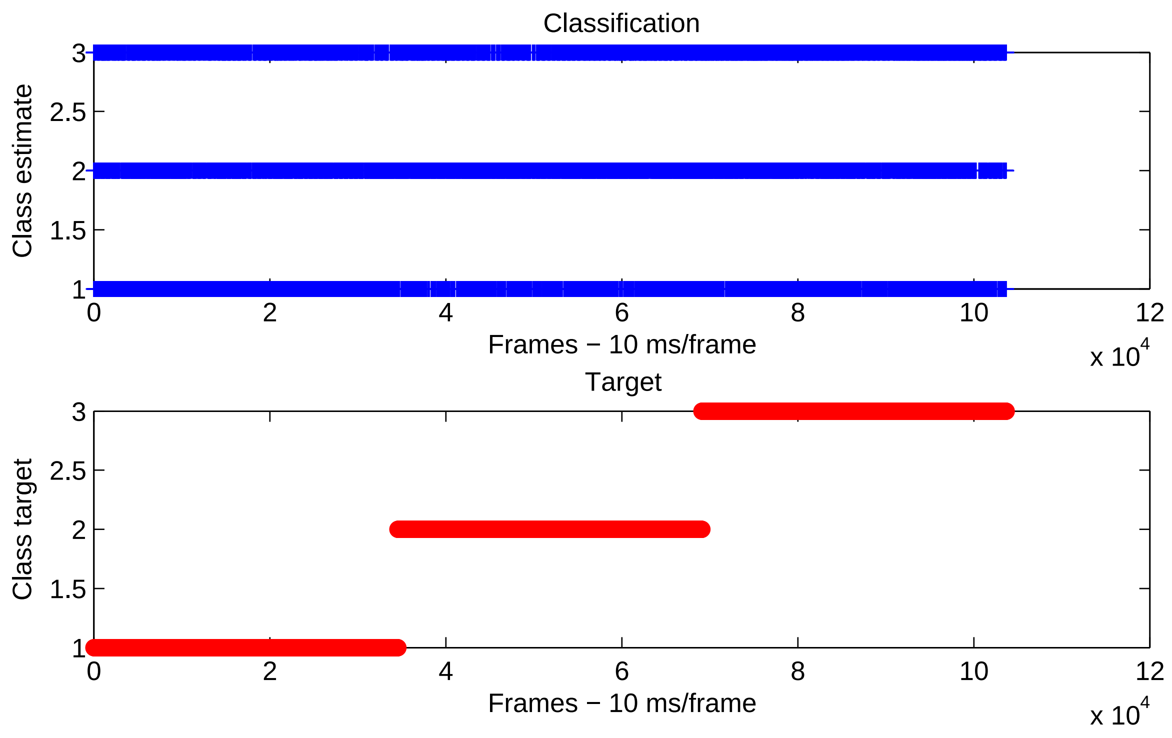

Figure 3.3: Results of using linear classifiers and ten digits spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	16346.0	7356.0	6220.0	54.6
Estimate Mose	8902.0	15435.0	7615.0	48.3
Estimate Simon	9311.0	11768.0	20724.0	49.6
Sensitivity [%]	47.3	44.7	60.0	50.6

Table 3.3: Confusion matrix - 10 digits

3.4 Discussion

The target vector and the estimated targets for the test of one digit is shown on figure 3.1. For two and ten digits the estimated targets is shown respectively 3.2 and 3.3. The confusion matrix (CM) is calculated using the MATLAB code. The CM of one digit is displayed on table 3.1, The overall accuracy of the linear classification is calculated to 55.9 %. The CM of two digits is displayed on table 3.2, The overall accuracy of the linear classification is calculated to 51.2 %. The CM of ten digits is displayed on table 3.3, The overall accuracy of the linear classification is calculated to 50.6 %. The CM of the classification show that the model has decreasing accuracy for higher numbers of digits. Generally the linear classifier is not optimal for speaker recognition, this also shows in the accuracy. The model is still over the accuracy of 33 %, which is a random classifier, but not as reliable as the other model, later in the rapport.

4 Dimensionality Reduction

4.1 Theory

Dimensionality reduction is used to reduce the number of variables of the feature extraction. There is two methods to make Dimensionality reduction, the first method is Fisher's linear discriminant model. Fisher's model takes a D-dimensional input vector $\mathbf{x}$ and project to one dimension bye the equation:

$$y = \mathbf{w}^T \mathbf{x}. \tag{4.1}$$

The projection down to one dimension leads to loss of information, and the data even if it was well separated in D-dimensions, can become overlapping when only viewed in one dimension. The other method is Principal component analysis (PCA), which can be used to make a dimensionality reduction of the dataset. PCA works bye orthogonal project the of the data onto a lower dimensional linear space. The new space is called principal subspace and are made so that the variance of the projected data is maximized.

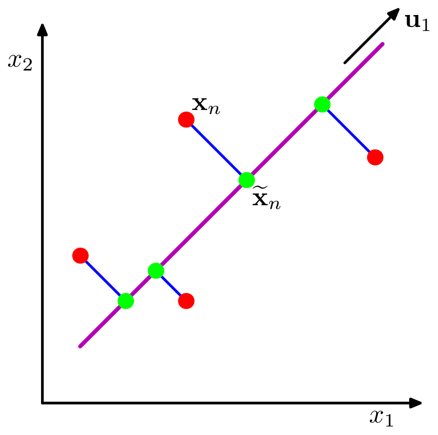

Figure 4.1: Principal component analysis orthogonal project the of the data onto a lower dimensional linear space, This is shown on the figure. The figure is borrowed from [4].

In this project PCA is used to make the dimensional reduction focusing on the maximum variance approach.

4.1.1 Maximum Variance

Consider a dataset of observations $\{\mathbf{x}_n\}$ with $n = 1, ..., N$, and the data vector is a Euclidean variable with dimension D. The dataset can be projected to lower dimensional space with dimension $M < D$ while maximizing the variance of the projected data. The projected space of dimension M is based on the eigenvectors

$\mathbf{u}_1, ..., \mathbf{u}_M$. Consider a projection onto $M = 1$ dimension, the projected space is spanned by the vector $\mathbf{u}_1$. Then each data point $\mathbf{x}_n$ is projected onto a scalar $\mathbf{u}_1^T \mathbf{x}_n$. The mean of the sample set is given by $\overline{\mathbf{x}} \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}_n$. The calculate the variance of the projected data is given by

$$\frac{1}{N} \sum_{n=1}^{N} \left(\mathbf{u}_1^T \mathbf{x}_n - \mathbf{u}_1^T \overline{\mathbf{x}} \right)^2 = \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 \tag{4.2}$$

Where $\mathbf{S}$ is the covariance matrix. Then maximize the projected variance $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$ with respect to $\mathbf{u}_1$. Which show that the $\mathbf{u}_1$ is an eigenvector of $\mathbf{S}$.

$$\mathbf{S}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1 \tag{4.3}$$

Multiplying by $\mathbf{u}_1$ on the left.

$$\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 = \lambda_1 \tag{4.4}$$

The maximum of the variance is $\mathbf{u}_1$ is set equal to the eigenvector having the largest eigenvalue λ_1. This eigenvector is called the first principal component. When M is set higher than 1, additional principal component can be defined by choosing each new direction that maximizes the projected variance. This can be used to remove the dimensions that have small eigenvalue and therefore little information.

4.2 Method

In this project PCA has been applied, mainly in an effort to improve the quality of the data by de-correlating it. It has not been the focus to try and reduce the dimensionality of the data, as real-time processing cost has not been a deciding factor. Subsequently, all analysis of classification methods has been done on data that has been de-correlated using PCA, but not reducing in dimensionality.

The effect og dimensionality reduction has though been studied, and is presented in this section

In order to gauge the effect of dimensionality of our data, we first analyse how much of the variance is contained in how many dimensions, and then look at the cost in accuracy of truncating dimensions of the de-correlated dataset.

4.3 Results

4.3.1 Distribution of variance:

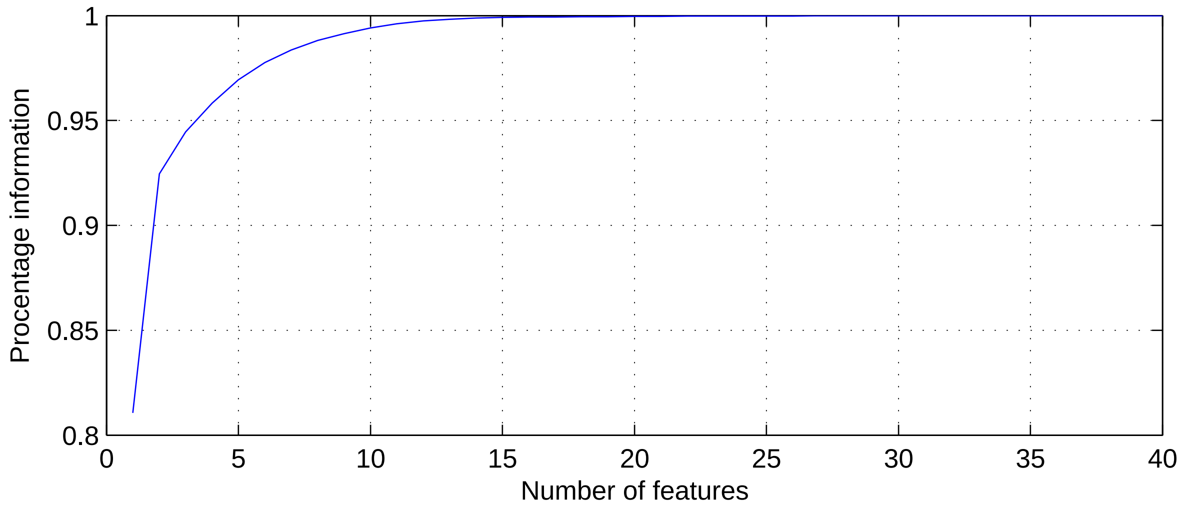

Figure 4.2: Cumulative distribution of variance (information) in dimensions sorted by highest eigenvalue

4.3.2 Cost of dimensionality reduction:

The cost of dimensionality reduction is studied for linear classifiers and for Gaussian Mixture Models.

Linear classifiers

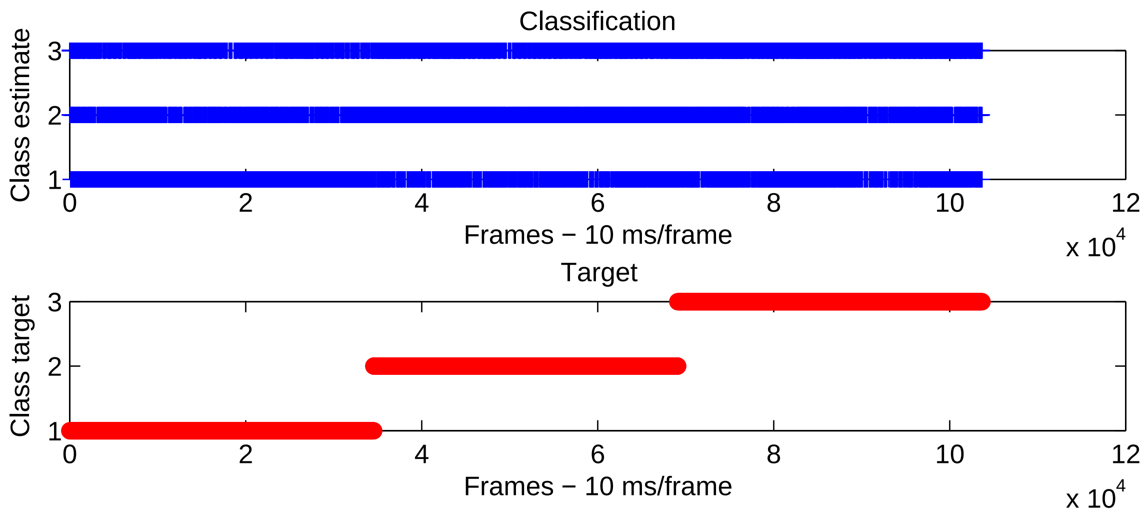

Figure 4.3: Results of using linear classifiers and ten digits spoken, with a truncated 10-dimensional feature space

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	16017.0	7631.0	6257.0	53.6
Estimate Mose	9131.0	15367.0	7215.0	48.5
Estimate Simon	9411.0	11561.0	21087.0	50.1
Sensitivity [%]	46.3	44.5	61.0	50.6

Table 4.1: Confusion matrix - Linear classifier, truncated to 10 dimensions

Gaussian Mixture Models

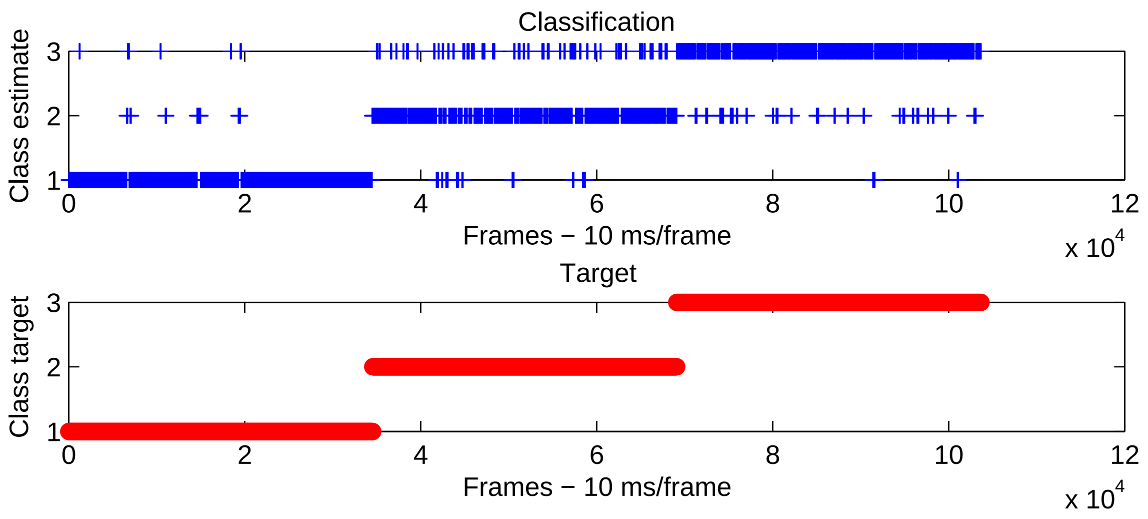

Figure 4.4: Results of using GMM and ten digits spoken, with a truncated 10-dimensional feature space

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	33000.0	1400.0	300.0	95.1
Estimate Mose	959.0	27341.0	3200.0	86.8
Estimate Simon	600.0	5818.0	31059.0	82.9
Sensitivity [%]	95.5	79.1	89.9	88.2

Table 4.2: Confusion matrix - GMM, truncated to 10 dimensions

4.4 Discussion

Figure 4.2 shows cumulative distribution of variance in the separate dimensions of the feature space. In this it is apparent most of the variance is contained within the first 5-10 dimensions. 94.9% at 5 dimensions and 99.4% at 10 dimensions. This does not necessarily imply that most of the information is contained within these, but is a good indicator of this.

The data shown in Figure 4.3 and Table 4.1 for linear classifiers and Figure 4.4 and 4.2 for GMM, should be compared with Figure 3.3 Table 3.3, Figure 8.7 and Table 8.3 respectively.

In this we see that truncating the feature space to one quarter of the dimensions (10) after PCA, has a very little effect on the overall accuracy of the classification. This is important, since it can allow the classification algorithms to run with a much lower computational load.

In this project, computational load has not been a major concern. For this reason and the reasons above it we have chosen not to truncate our feature space.

5 Probabilistic Generative Models

In this part of the appendix, describes the Probabilistic Generative models (PGM) and how it is used on the dataset.

5.1 Theory

The model is a multivariate normal distribution, which means that each class corresponds to a unique distribution.

$$p(\mathbf{x}|C_k) = \mathcal{N}(\mathbf{x}; \mu_k,\ \Sigma_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\mathbf{\Sigma}|^{1/2}} \texttt{exp} \left\{ -\frac{1}{2}(\mathbf{x} - \mu_k)^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mu_k) \right\} \tag{5.1}$$

The probability distribution is used in addition to the prior probability of each class, This enables the possibility to calculate the posterior probability of a class given a feature vector from Bayes theorem.

$$p(C_k|\mathbf{x}) = \frac{p(\mathbf{x}|C_k)p(C_k)}{\sum_j p(\mathbf{x}|C_j)p(C_j)} \tag{5.2}$$

The prior probabilities for each class are approximated to be equal, to indicate that class k is as frequent as class j, which leads to a simplification of Bayes theorem.

$$p(C_k) = p(C_j) = \alpha \Longrightarrow p(C_k|\mathbf{x}) = \frac{\alpha \cdot p(\mathbf{x}|C_k)}{\alpha \cdot \sum_j p(\mathbf{x}|C_j)} = \frac{p(\mathbf{x}|C_k)}{\sum_j p(\mathbf{x}|C_j)} \tag{5.3}$$

5.2 Method

To make the generative models the mean vector and covariance matrix for each class was calculated using all the training data for the respective classes.

$$\boldsymbol{\mu}_i = \frac{1}{N} \sum_{j=1}^{N} \mathbf{x}_{kj} \tag{5.4}$$

$$\mathbf{\Sigma}_k = \frac{1}{N} \sum_{j=1}^{N} (\mathbf{x}_{kj} - \boldsymbol{\mu}_k) \cdot (\mathbf{x}_{kj} - \boldsymbol{\mu}_k) \tag{5.5}$$

The class-conditional densities, for all samples and for all speaker models was then calculated using (5.1). Then the posterior probability for all classes are calculated using Bayes' theorem (5.2) and assuming uniform class priors. Finally the samples were classified by comparing posterior probabilities for all classes and selecting the greatest.

5.3 Results

5.3.1 1 digit:

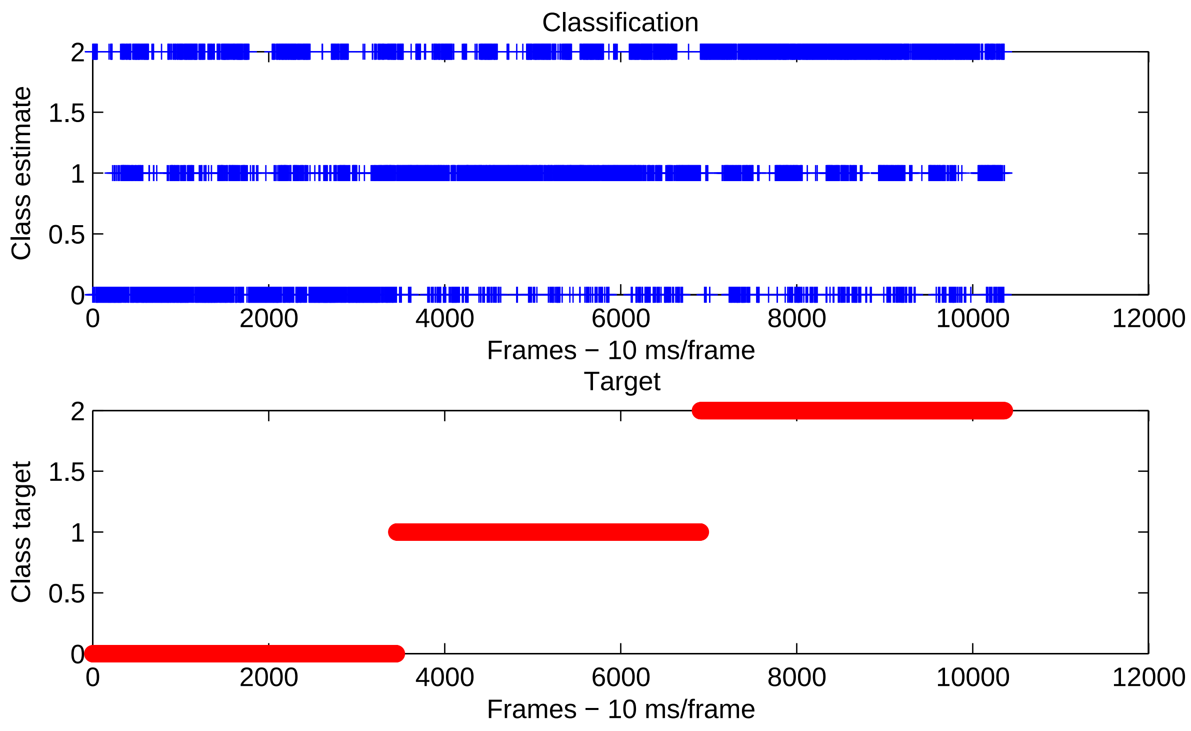

Figure 5.1: Results of using PGM classifiers and one digit spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	2027.0	313.0	391.0	74.2
Estimate Mose	569.0	2199.0	794.0	61.7
Estimate Simon	858.0	942.0	2269.0	55.8
Sensitivity [%]	58.7	63.7	65.7	62.7

Table 5.1: Confusion matrix - 1 digit

5.3.2 2 digits:

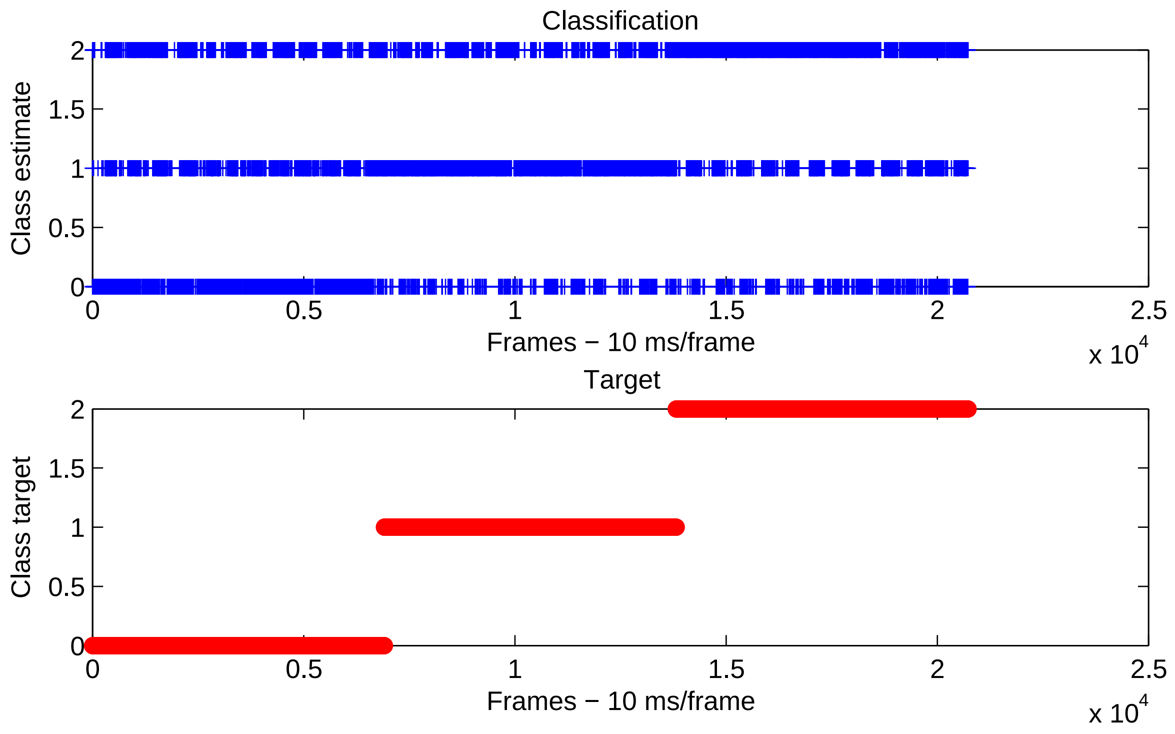

Figure 5.2: Results of using PGM classifiers and one digit spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	3471.0	585.0	1051.0	68.0
Estimate Mose	1567.0	4213.0	1500.0	57.9
Estimate Simon	1872.0	2112.0	4359.0	52.2
Sensitivity [%]	50.2	61.0	63.1	58.1

Table 5.2: Confusion matrix - 2 digits

5.3.3 10 digits:

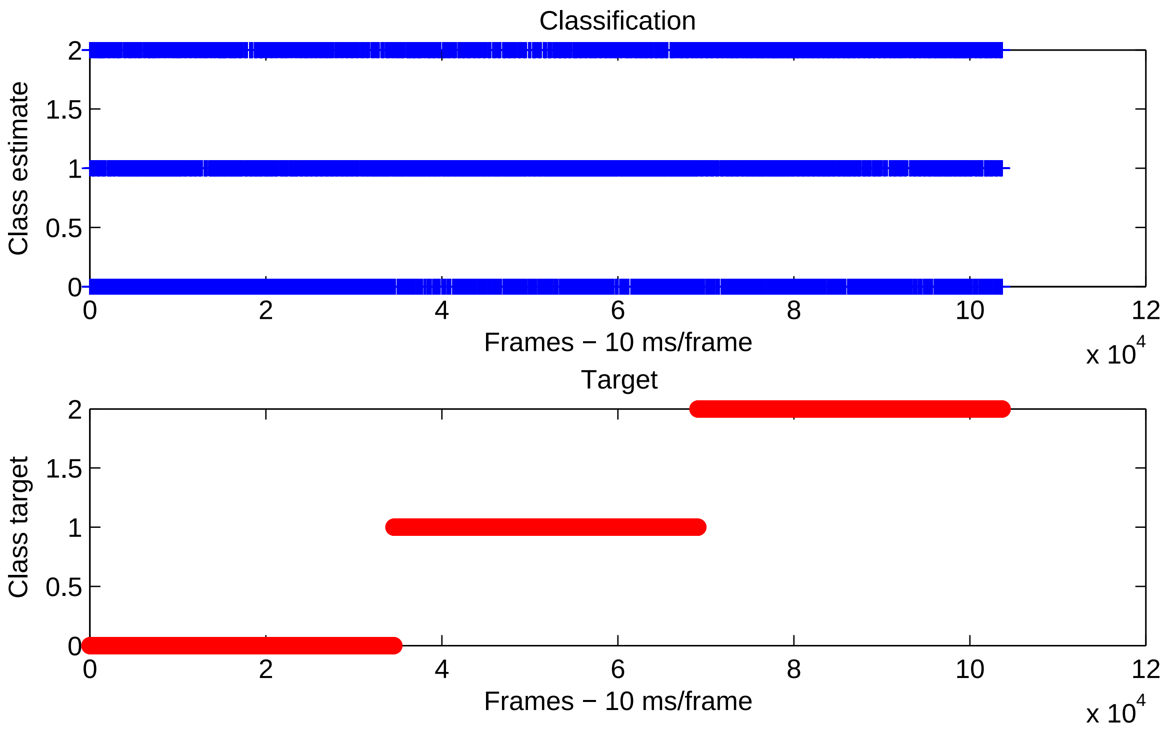

Figure 5.3: Results of using PGM classifiers and ten digits spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	18641.0	5089.0	5091.0	64.7
Estimate Mose	9440.0	20760.0	9570.0	52.2
Estimate Simon	6478.0	8710.0	19898.0	56.7
Sensitivity [%]	53.9	60.1	57.6	57.2

Table 5.3: Confusion matrix - 10 digits

5.4 Discussion

Overall the accuracy of PGM ranging from 62.7% for single digit [1] to 57.2% for all ten digits [2], is significant relative to a random classification of 33%. It is, though, not good enough for reliable classification.

This is to be expected, as modelling all the sounds in a word or digit precisely, let alone 10 of them, with a single multivariate Gaussian is not feasible.

[1](See table 5.1)
[2](See table 5.3)

6 Artificial Neural Networks

6.1 Theory

Artificial Neural Networks (ANN) also called multilayer perceptron, can be used as a model for pattern recognition. The neural network model is a nonlinear function from a set of input variables $\{x_i\}$ transformed to a set of output variables $\{y_k\}$ controlled by the weight vector $\mathbf{w}$, which is made of adjustable parameters. The number of hidden units between the input and the output can be adjusted to the dataset, the process is described in the method.

The overall network function for a two layer neural network, takes the form

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{6.1}$$

The superscript (1) and (2) indicates which layer the parameter is in, also shown on the parameters in figure 6.1. Both $\sigma(\cdot)$ and $h(\cdot)$ are activation functions, which can be chosen to be either the logistic sigmoid or the tanh function. The part with $h(...)$ is in the context of ANN called the hidden units. D is the number of inputs, M is the number of hidden units and K is the number of outputs.

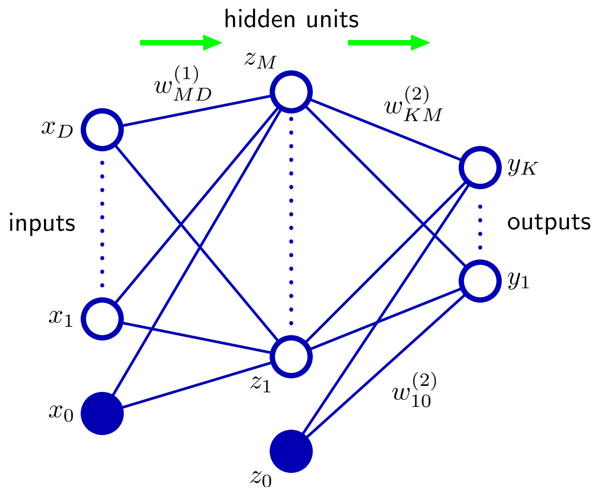

Figure 6.1: Diagram of the two layer neural network. The nodes in the figure represents the input, hidden and output variable. The links between the nodes are the weight parameters. The figure is borrowed from [4]

6.2 Method

In both the training and test phases the Netlab Toolbox [5] was used in MATLAB. In the training of the neural network for a multi-class classification problem the following error-function is minimized.

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2 + \lambda|\mathbf{w}^T \cdot \mathbf{w}| \tag{6.2}$$

Where $\mathbf{y}(\mathbf{x}_n, \mathbf{w})$ is the network function with the parameters and biases in $\mathbf{w}$ and the target vector, $\mathbf{t}_n$, is written as a one-of-K-coding. By minimizing the error-function the parameter that are best for classifying this type of data is fund. The parameter includes number hidden units and the α, which is the inverse variance of the Gaussian noise. The dataset that was used on the error-function, was the single digit data. The reason for this, is the vast time it takes to minimizing the error-function, therefore the smallest dataset is used. This can however make a small error in the number of hidden units optimal for the big dataset of ten digits.

The ANN was trained for the following number of hidden values: [10 20 50 100 200 300]. For each number of hidden values the training was done for 8 different values of α, uniformly space between 0 and 1. for each value of alpha the training was tried 8 times and the mean and variance of the error was recorded. The results are visible below in Figure 6.2.

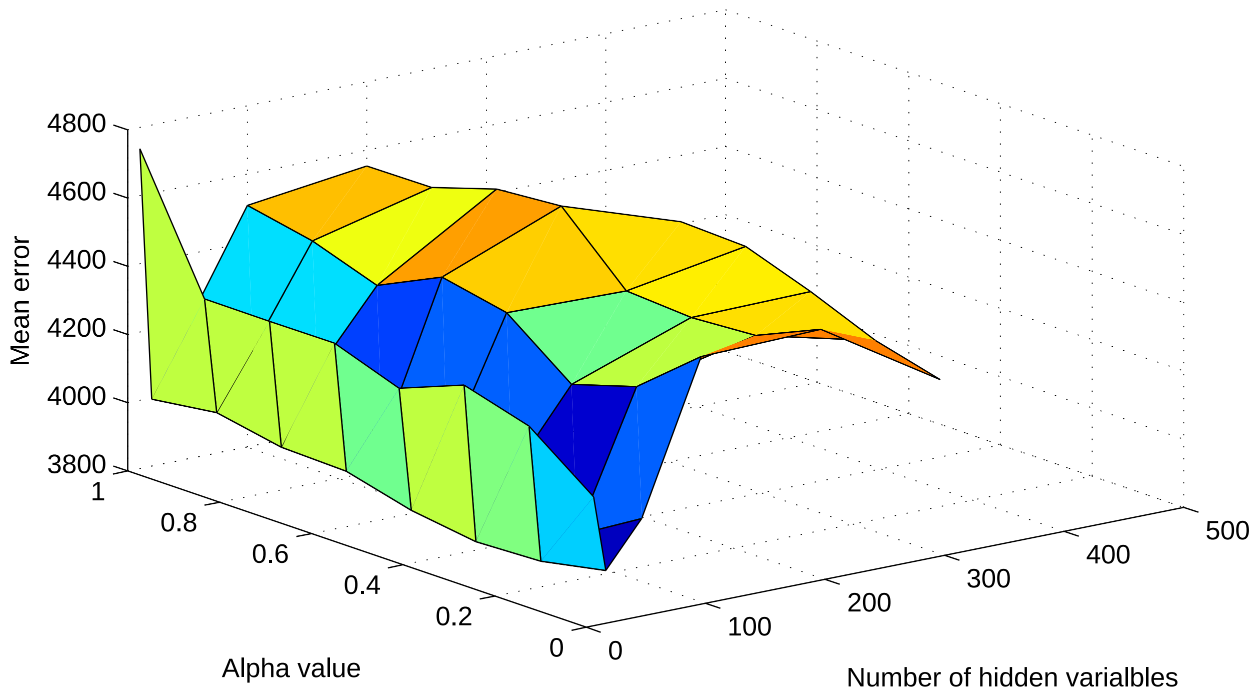

Figure 6.2: Results of parametric analysis. Mean error displayed.

The number of hidden units was found to be optimal at 30, with an α of 0.30. This is used for all of the datasets.

6.3 Results

Using ANN has been tried with different data complexities. Most notably, with speakers uttering the same single digit ("*ZERO*"), two different digits ("*ZERO*" and "*ONE*"), and ten different digits ("*ZERO*", "*ONE*" through "*NINE*").

6.3.1 Single digit:

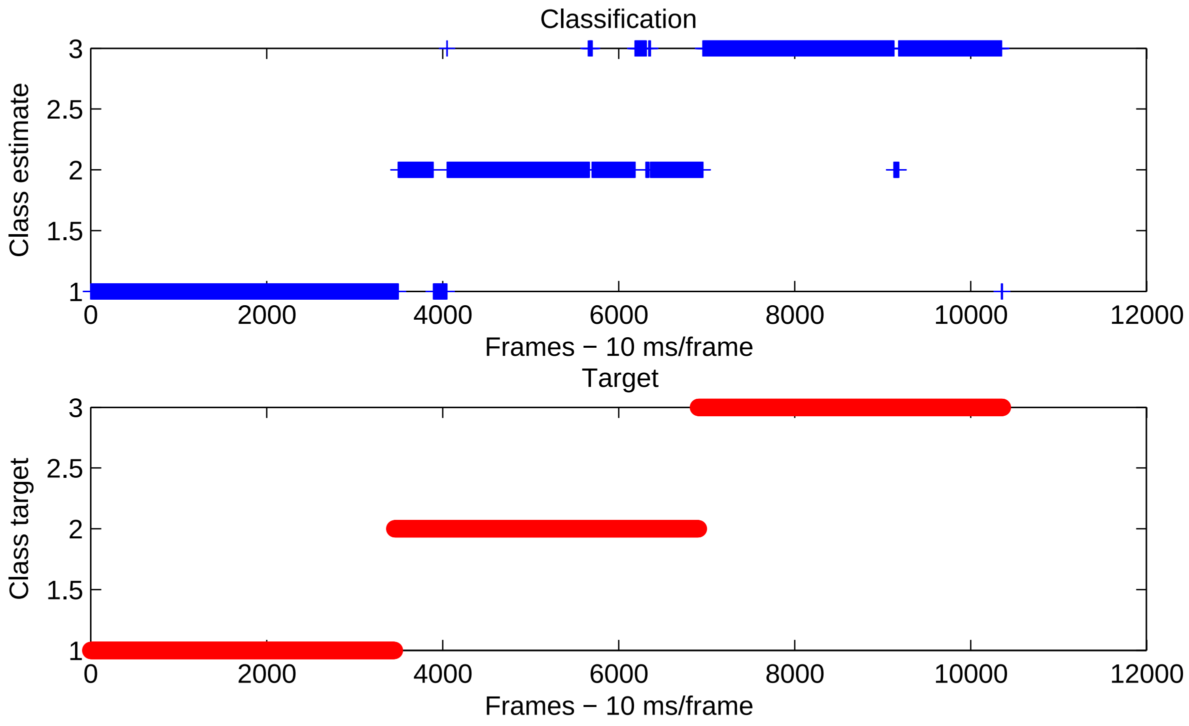

Figure 6.3: Results of using ANN with 3 speakers, 30 hidden variables and 1 digit spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	3454.0	196.0	11.0	94.3
Estimate Mose	0.0	3077.0	104.0	96.7
Estimate Simon	0.0	181.0	3339.0	94.9
Sensitivity [%]	100.0	89.1	96.7	95.3

Table 6.1: Confusion matrix - 1 digit

6.3.2 Two digits:

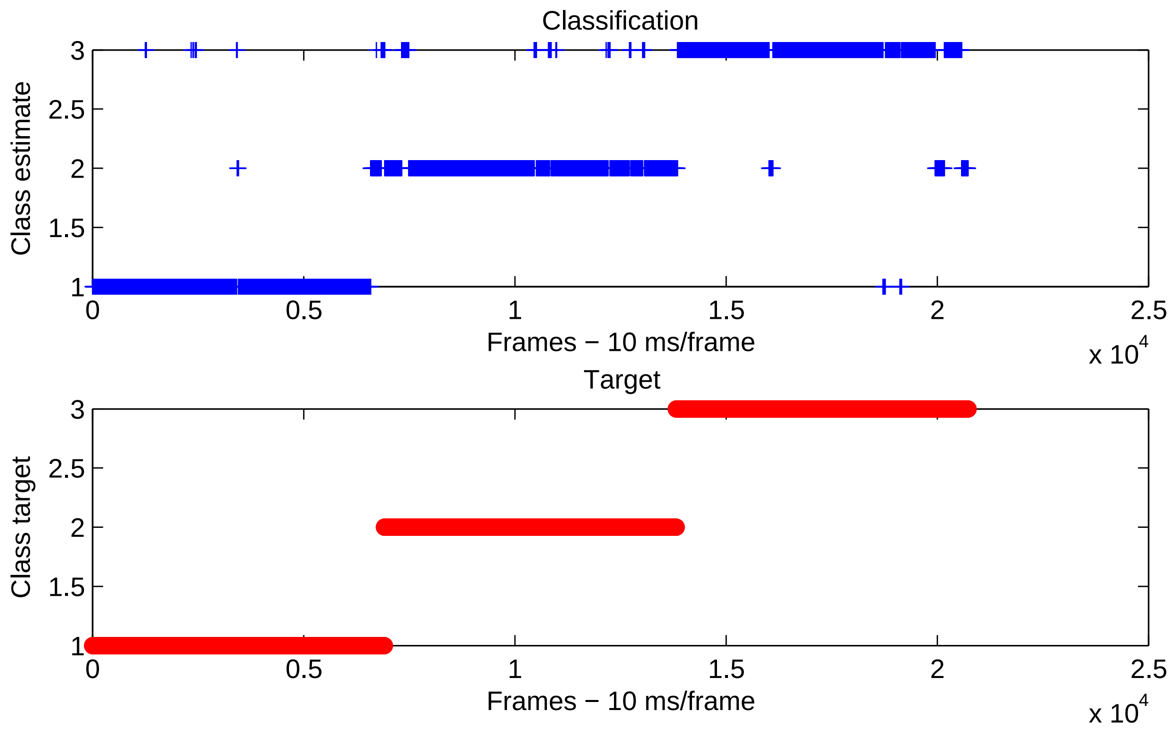

Figure 6.4: Results of using ANN with 3 speakers, 30 hidden variables and 2 digits spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	6486.0	0.0	93.0	98.6
Estimate Mose	276.0	6485.0	492.0	89.4
Estimate Simon	148.0	425.0	6325.0	91.7
Sensitivity [%]	93.9	93.8	91.5	93.1

Table 6.2: Confusion matrix - 2 digits

6.3.3 Ten digits:

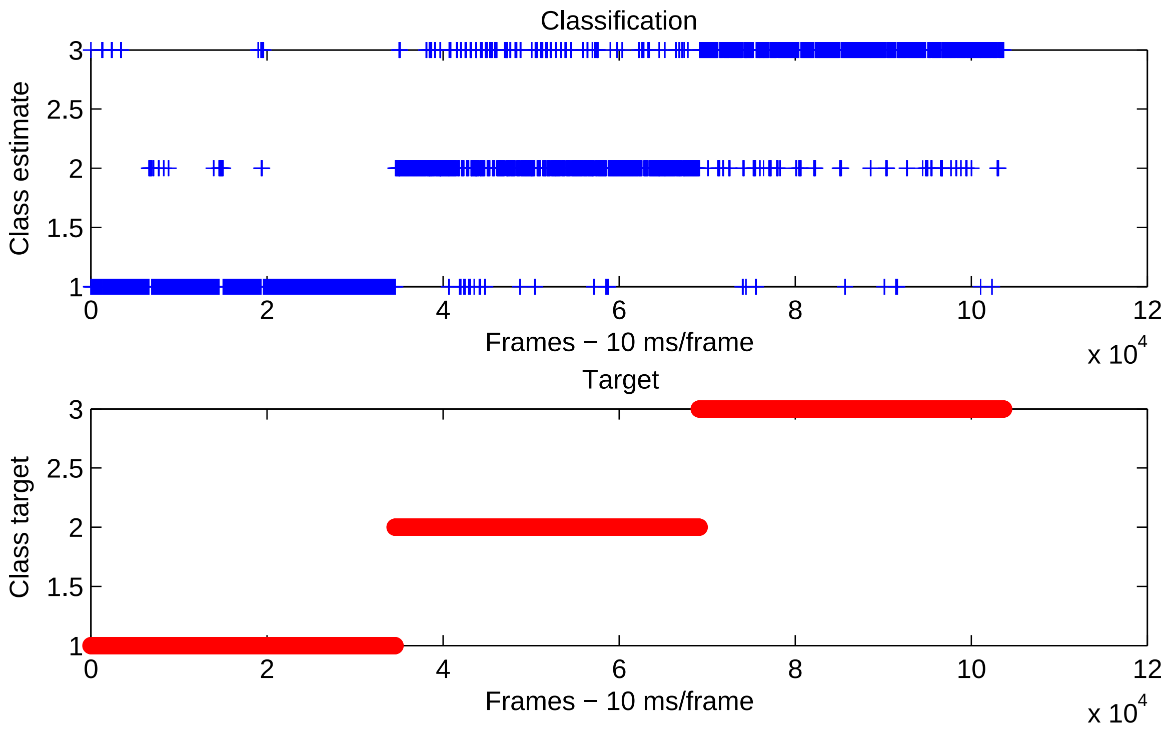

Figure 6.5: Results of using ANN with 3 speakers, 30 hidden variables and 10 digits spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	32915.0	1186.0	528.0	95.1
Estimate Mose	1245.0	28807.0	3100.0	86.9
Estimate Simon	399.0	4566.0	30931.0	86.2
Sensitivity [%]	95.2	83.4	89.5	89.4

Table 6.3: Confusion matrix - 10 digits

6.4 Discussion

In this project there the three test dataset. The CM of one digit is displayed on table 6.1, The overall accuracy of the linear classification is calculated to 95.3 %. The CM of two digits is displayed on table 6.2, The overall accuracy of the linear classification is calculated to 93.1 %. The CM of ten digits is displayed on table 6.3, The overall accuracy of the linear classification is calculated to 89.4 %. The CM of the classification show that the model has decreasing accuracy for higher numbers of digits. This probably because of the number of hidden units is calculated for the dataset of one digit. The result could be optimized for larger number of digits, if the minimization of the error-function was used on the other dataset. This was not done because the process is very time consuming.

The overall accuracy of the ANN model is very good, and the model can be used as an reliable speaker recognition classifier. In this project the best model to recognise a speaker is the Artificial Neural Networks, with the highest overall accuracy.

7 K-means clustering

K-means clustering is an unsupervised machine learning algorithm for finding sub populations in unlabelled data. It has not been applied to this case, but is reviewed for sake of completeness and to help understanding of Gaussian Mixture Models (See chapter 8)

7.1 Theory

K-means basically works by iteratively estimating K cluster means and assigning responsibility according to euclidean distance.

7.1.1 The K-means algorithm

1. Choose the number, K, of clusters to fit.
2. Pick K random guesses of mean values μ_j or mean vector $\boldsymbol{\mu}_k$ for multi dimensional data.
3. Assign responsibility of all data point by smallest euclidean distance to cluster mean.

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\mathbf{x}_n - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

   Where r_{nk} is 1 if cluster k is responsible for $\mathbf{x}_n$

4. Estimate new cluster means from cluster assignments

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk}\mathbf{x}_n}{\sum_n r_{nk}} \tag{7.2}$$

5. Repeat from 3. until stopping criteria is met e.g. no change in cluster assignments or change in cluster means below pre-set threshold.

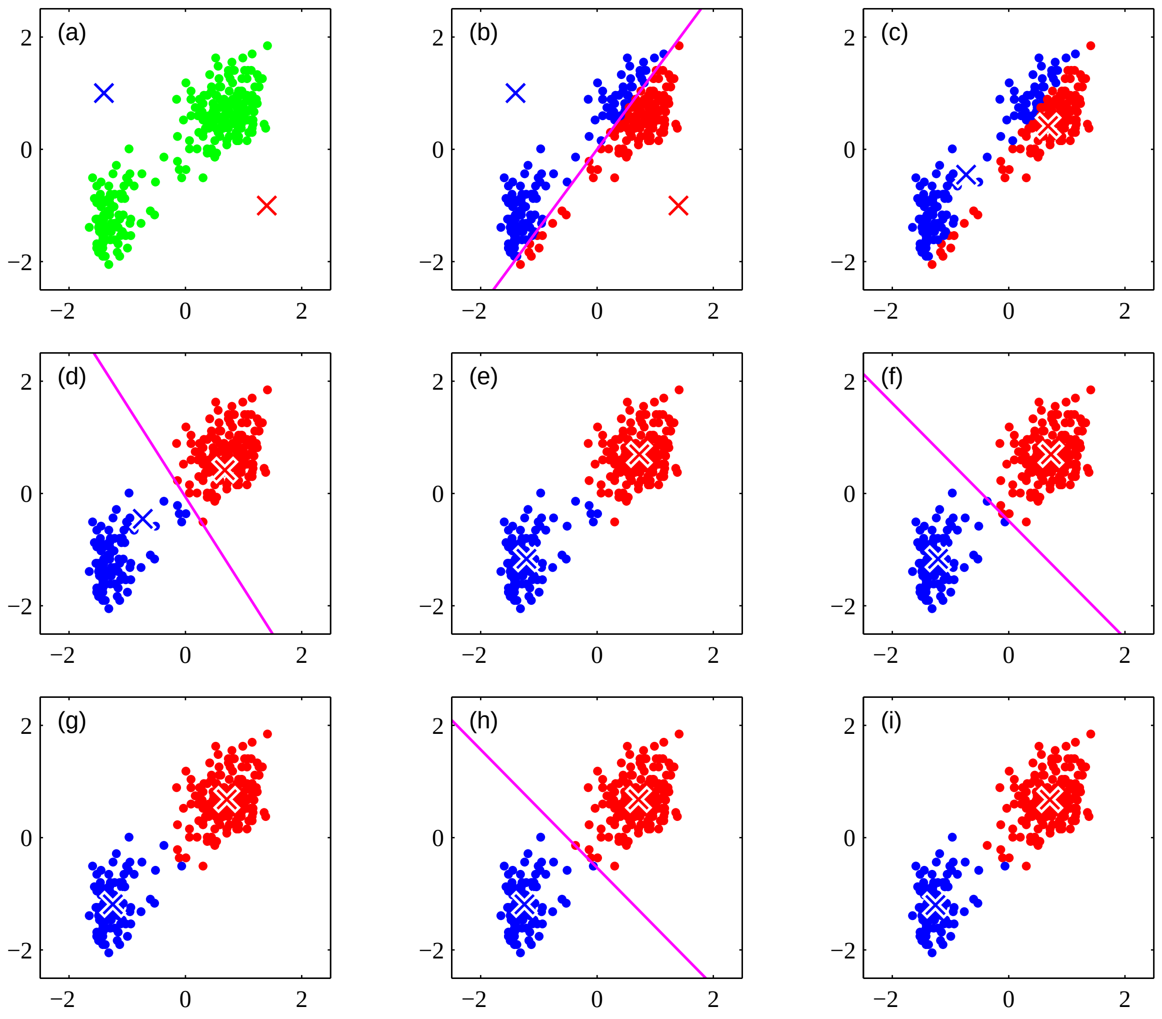

Figure 7.1: Four iterations of K-means on the Old Faithful data set

7.2 Discussion

In this project K-means could have been used to make a sort of mixture model of each speaker, by fitting a number of clusters to the different unique sounds the individual speaker makes and then for a series of samples evaluating which speaker is most likely to have emitted the sounds.

It has been elected not to try to apply K-means to this project as it does not fit this application very well, as opposed like other more advanced methods like Artificial Neural Networks (Chapter 6) or Gaussian Mixture Models (Chapter 8).

8 Gaussian Mixture Models

8.1 Theory

Gaussian Mixture Models (GMM) is a way of finding and describing sub-populations in clusters of data points. It is done by fitting a specified number of Gaussian distributions to a population of data points. Each distribution is a component of the model. The individual data points are then arranged into clusters based on which model component is most likely given the observed data point.

In this context the data points are features in a multidimensional feature space. Hence the distributions used for the GMM is multivariate Gaussians.

The clustering on basis of likelihood makes GMMs more robust than e.g. K-means, where data points are clustered similarly, but simply on the basis of Euclidean distance to the center of a model. This is because treating the components like Gaussians with means and variances instead spheres with uniform probability, gives a more nuanced picture of the strength of a data point's relationship to a model component. It also gives rise to the notion of soft relations to clusters or data points related to more than one cluster.

When fitting the GMM the goal is to maximize the overall likelihood of the model for the entire population. Firstly it is necessary to determine the number of distribution components the data should be fitted to. The number of distributions needed in a GMM greatly depends on the nature and origin(s) of data. This, however, does not mean that if a mixture model fits data well with a specific amount of distributions, then this is the actual number of sub-populations, or sources if you will. Multiple sub-populations could be grouped together, or likewise sub-populations could be split, due to under/over-fitting. For this reason experimentation is necessary to establish the optimal number of distributions needed to model data.

8.1.1 The EM Algorithm

The distributions of the mixture model are fitted to data by iteratively employing Expectation Maximization (EM). This is done by either selecting an arbitrary guess of the means and variances of the model as a starting point or initializing the means with a few iterations of K-means, and setting the covariance matrices to the identity matrix. The latter is often used because even though GMM generally is more precise than K-means, it is also much more computationally heavy. Therefore by initializing with K-means, a relatively light algorithm, the EM will converge on a good model faster.

A visual example of this convergence can be shown by applying GMM to dataset of eruption time vs. waiting time of the Old Faithful geyser in Yellowstone National Park, WY, USA. The data has been normalized for simplicity.

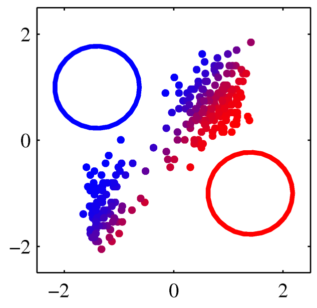

Figure 8.1: Old Faithful data. GMM stared at an abitrary point

In Figure 8.1 above the EM has just started from an arbitrary starting point. Data shows two apparent clusters, so the model is generated with two components. The data points are colored by association with the model components. Note the fading in colors showing strength of relation to model component. At this point the GMM does not fit data very well.

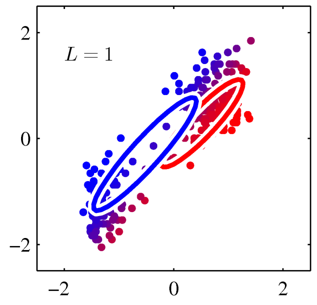

Figure 8.2: Old Faithful data. GMM after one iteration of EM

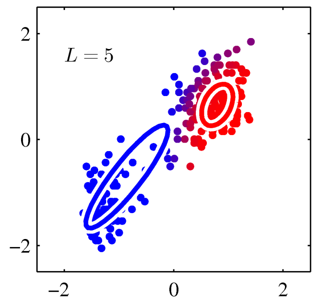

Figure 8.3: Old Faithful data. GMM after five iterations of EM

After five iterations the EM has honed in on the centers of the two clusters (Figure 8.3). Note that K-means could have reached roughly this point in as many iterations, but at much less computational cost.

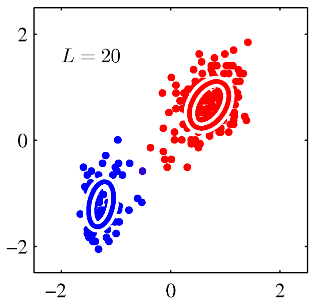

Figure 8.4: Old Faithful data. GMM after 20 iterations of EM

As seen in Figure 8.4 above the model has converged on a well-fitting model after 20 iterations. This means that further iterations would be superfluous, as they would yield only negligible increases in overall likelihood of the GMM.

The specific procedure for estimating GMM parameters using EM is as follows [4]:

Initialization:

Choose initial estimates for model parameters $\pi_k, \mu_k, \mathbf{\Sigma}_k$.

- k is the component number out of K.
- π_k is the weight of the kth component.
- μ_k is the mean of the kth component.
- $\mathbf{\Sigma}_k$ is the covariance matrix of the kth component.

Compute the initial log-likelihood og the model

$$\ln p\,(X|\mu, \mathbf{\Sigma}, \pi) = \sum_{n=1}^{N} \ln \sum_{k=1}^{N} \pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \mathbf{\Sigma}_k) \tag{8.1}$$

E-step:

Calculate the probability of each point in each component in order to assing responsibility

$$\gamma_{nk} = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \mathbf{\Sigma}_k)}{\sum_{j=1}^{K} \pi_n \mathcal{N}(\mathbf{x}_n|\mu_j, \mathbf{\Sigma}_j)} \tag{8.2}$$

M-step:

Estimate new guesses for $\pi_k, \mu_k, \mathbf{\Sigma}_k$.

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} \mathbf{x}_k \tag{8.3}$$

$$\mathbf{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_{nk} (\mathbf{x}_k - \mu_k^{new})(\mathbf{x}_k - \mu_k^{new})^T \tag{8.4}$$

$$\pi_k^{new} = \frac{N_k}{N} \tag{8.5}$$

Convergence check:

Recalculate log likelihood using Equation 8.1. If the log likelihood has not changed more than some predetermined threshold stop iteration. Otherwise continue from E-step.

8.2 Method

The way GMM was applied in this project was, in simple, to generate a single GMM for each speaker using the training data. Then the test set was classified by summing the log-likelihoods for each GMM for a number of sequential frames and then selecting the speaker with the highest sum of log-likelihoods. This approach adds a measure of supervision to an otherwise unsupervised technique. Further, because classification is done using a set of sequential frames, a temporal element is added.

8.2.1 Training

A GMM was trained for every speaker. The feature data (both training and test data) has been subjected to PCA, in order to project the data onto a basis that aligned the variance along the dimensions without removing any dimensions. Then, for each speaker, a GMM is fitted to the respective speakers' training data, using MATLAB's Statistics Toolbox and the following parameters:

- Number of components: *8*
- Covariance matrix: *Full*
- Initialization: *Random*

8.2.2 Classification

Classification is done on the test data is done on super frames of 100 sequential frames. Each frame assumed to be independent. For this reason the probability of on super frame given a specific model equals the product of the probabilities for each frame.

$$p(\mathbf{x}_{n..n+100}|C_k) = \prod_{i=n}^{n+100} p(\mathbf{x}_i|C_k) \tag{8.6}$$

Products of many likelihoods have a tendency to result in very low numbers. Also multiplication is usually more computational intensive than addition. So in order to speed up calculations and alleviate numerical instability due to underflow the negative log-likelihoods are used.

$$p(\mathbf{x}_{n..n+100}|C_k) = \sum_{i=n}^{n+100} -\log(p(\mathbf{x}_i|C_k)) \tag{8.7}$$

The model with the largest sum is chosen as winner.

$$\text{class} = \arg\max_j \left\{ \sum_{i=n}^{n+100} -\log(p(\mathbf{x}_i|C_j)) \right\} \tag{8.8}$$

8.3 Results

Using GMM has been tried with different data complexities. Most notably, with speakers uttering the same single digit (”*ZERO*”), two different digits (”*ZERO*” and ”*ONE*”), and ten different digits (”*ZERO*”, ”*ONE*” through ”*NINE*”).

8.3.1 Single digit:

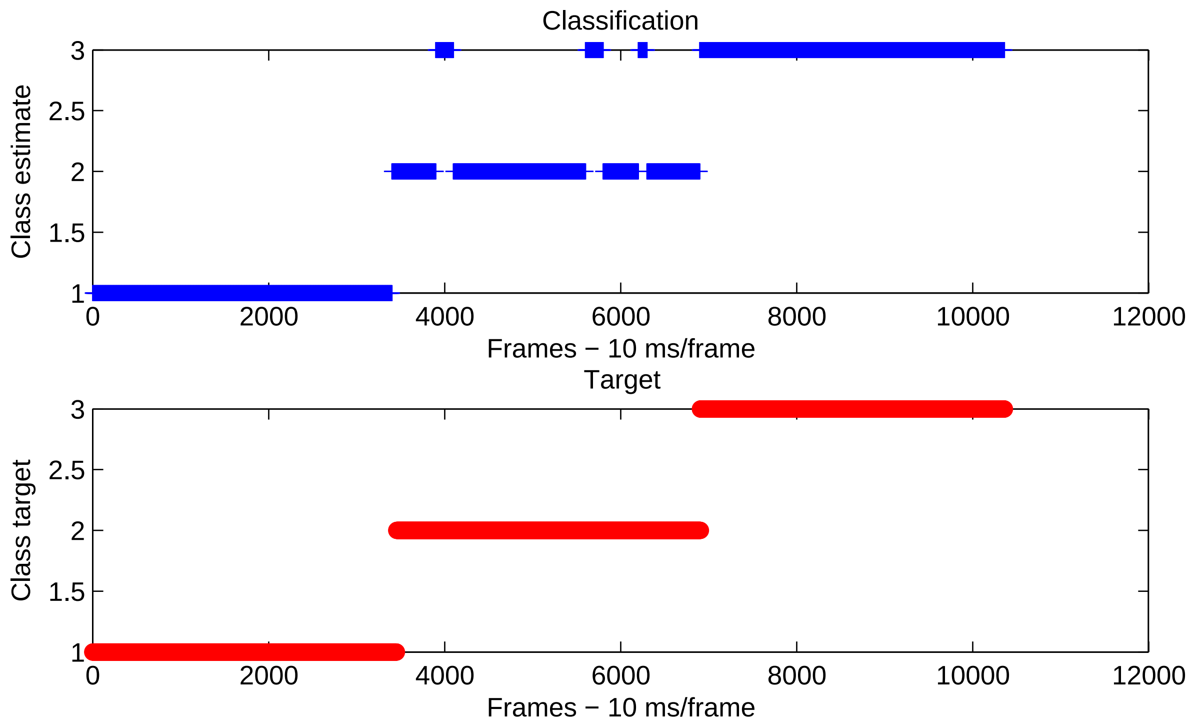

Figure 8.5: Results of using GMM with 3 speakers, 8 centers per model and 1 digit spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	3400.0	0.0	0.0	100.0
Estimate Mose	54.0	2946.0	0.0	98.2
Estimate Simon	0.0	508.0	3454.0	87.2
Sensitivity [%]	98.4	85.3	100.0	94.6

Table 8.1: Confusion matrix - 1 digit

8.3.2 Two digits:

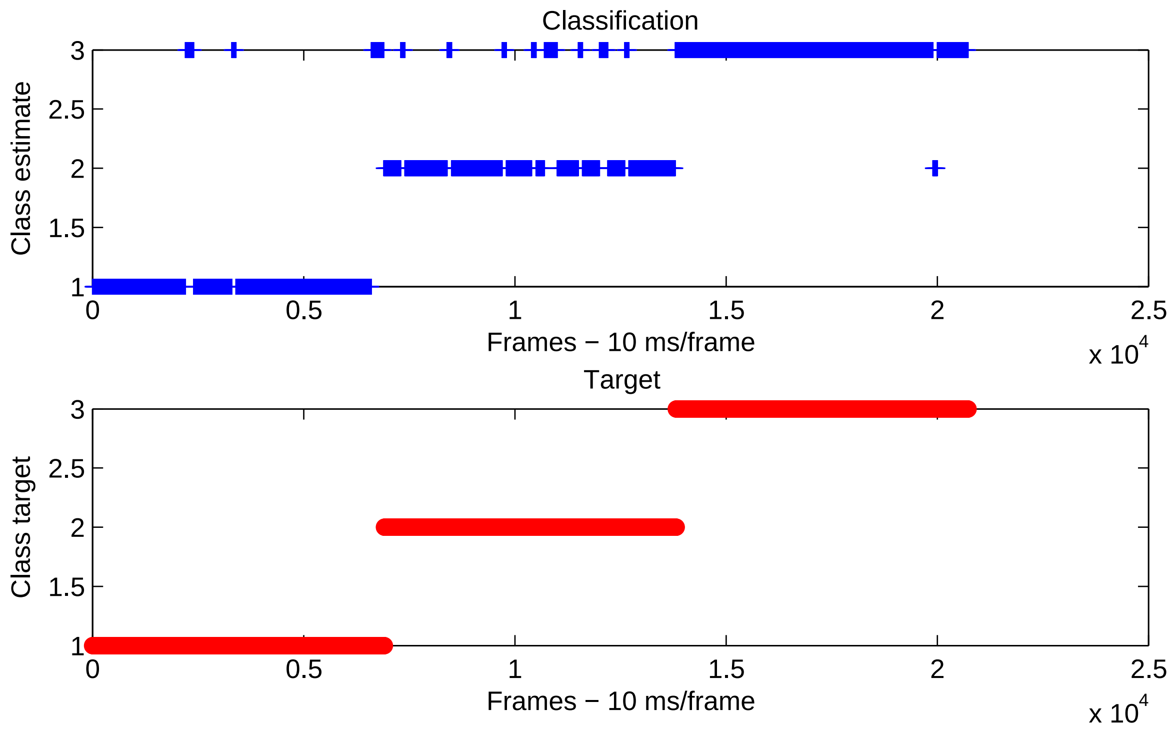

Figure 8.6: Results of using GMM with 3 speakers, 8 centers per model and 2 digits spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	6300.0	0.0	0.0	100.0
Estimate Mose	10.0	5790.0	100.0	98.1
Estimate Simon	600.0	1120.0	6810.0	79.8
Sensitivity [%]	91.2	83.8	98.6	91.2

Table 8.2: Confusion matrix - 2 digits

8.3.3 Ten digits:

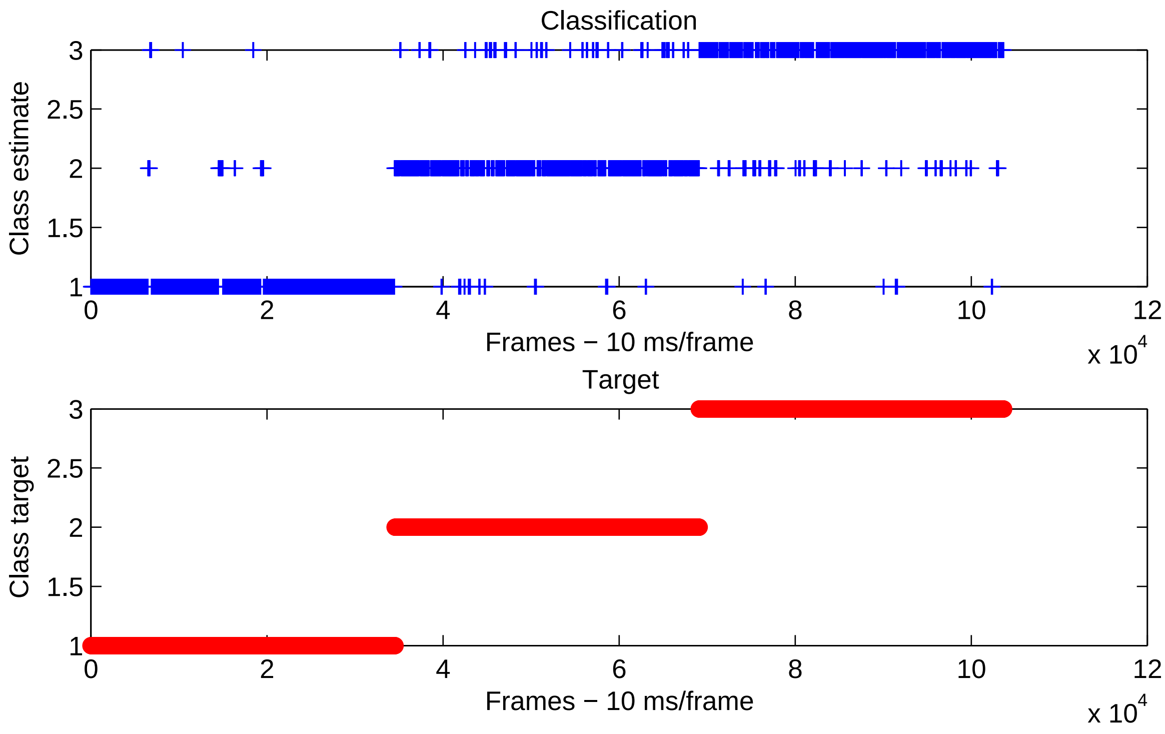

Figure 8.7: Results of using GMM with 3 speakers, 8 centers per model and 10 digits spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	33000.0	1300.0	600.0	94.6
Estimate Mose	1159.0	29341.0	3900.0	85.3
Estimate Simon	400.0	3918.0	30059.0	87.4
Sensitivity [%]	95.5	84.9	87.0	89.1

Table 8.3: Confusion matrix - 10 digits

8.4 Discussion

In this project there the three test dataset. The CM of one digit is displayed on table 8.1, The overall accuracy of the linear classification is calculated to 94.6 %. The CM of two digits is displayed on table 8.2, The overall accuracy of the linear classification is calculated to 91.2 %. The CM of ten digits is displayed on table 8.3, The overall accuracy of the linear classification is calculated to 89.1 %. The CM of the classification show that the model has decreasing accuracy for higher numbers of digits. The reason of this decreasing accuracy is the rising complexity of the dataset when the number of digits grow. The GMM model gives a very good result comparable with that of the ANN model. concluding that the GMM model can be used as an reliable speaker recognition classifier. The GMM model could be optimized with better fits, if the calculations had more computer power.

9 Support Vector Machines

9.1 Theory

In this section of the appendix the soft margin support vector machine (SVM) is described. The SVM is a binary classifier, which is a problem with multi class. The solution for multiclass classification is using a one-vs.-one setup. With only three classes this is achieved with relative ease. The integrated MATLAB statistic toolbox was used to process the data in the SVM case.

SVM classify by making a decision boundary that maximizes the margin γ. The margin is the perpendicular distance from the decision boundary to the closest feature point. The standard SVM have a linear decision boundary, which classifies by assigning the new data point to either class $t = 1$ or $t = -1$. The decision function for a standard SVM for a test point x_{new} given by

$$t_{new} = \mathtt{sign}(\mathbf{w}^T \mathbf{x}_{new} + b) \tag{9.1}$$

The parameter vector $\mathbf{w}$ is found by maximizing the margin or minimizing the length of the parameter vector, because of the inverse relationship $\gamma = \frac{1}{\|\mathbf{w}\|}$. Moving one single data point can have a huge influence on the position of the decision boundary, The reason is in the constrain, taken from [6].

$$\mathbf{w}^* = \mathtt{argmin}_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w}, \qquad t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \tag{9.2}$$

This however means that all the training features have sit on the correct side of the decision boundary. This way of classifying is called hard margin SVM, which can't handle overlapping classes. Therefore a slacken of the constrains is introduced $\xi_n \geq 0$ along with a regularization parameter C, This is called soft margin SVM

$$\mathbf{w}^* = \mathtt{argmin}_{\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N} \xi_n, \qquad t_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n \tag{9.3}$$

This optimization problem can be rewritten into the following quadratic problem by using Lagrange multipliers

$$\alpha^* = \mathtt{argmin}_{\alpha} \sum_{n=1}^{N} \alpha - \frac{1}{2}\sum_{n,m-1}^{N} \alpha_n\alpha_m t_n t_m \mathbf{x}_n^T\mathbf{x}_m, \qquad \sum_{n=1}^{N}\alpha_n t_n = 0, \qquad 0 \leq \alpha_n \leq C \tag{9.4}$$

The influence of each training point in the decision boundary is proportional to α_n. After the training process, the classification can be done by using the following expression, where $\mathbf{x}_n$ are the support-vectors, and α_n is their weights.

$$t_{new} = \mathtt{sign}(\mathbf{w}^T\mathbf{x}_{new} + b) = \mathtt{sign}\left(\sum_{n=1}^{N} \alpha_n t_n \mathbf{x}_n^T \mathbf{x}_{new} + b\right) \tag{9.5}$$

Instead of using a linear decision boundary, different kernels can be used to make other decision boundary. There are a linear kernel $\mathbf{x}_n^T\mathbf{x}_{new}$, polynomial kernel $(\gamma\mathbf{x}_n^T\mathbf{x}_{new})^d$, $\gamma > 0$. The kernel used in this project is the radial basis function $\texttt{exp}(-\gamma\|\mathbf{x}_n^T - \mathbf{x}_{new}\|^2)$, $\gamma > 0$, also called the Gaussian kernel.

$$t_{new} = \texttt{sign}\left(\sum_{n=1}^{N} \alpha_n t_n k(\mathbf{x}_n, \mathbf{x}_{new}) + b\right) \tag{9.6}$$

To expand this binary classifier to being able to process multiclass classification a one-vs.-one setup is used. This means that each of the classes is trained against all of the other classes, this results in $k(k-1)/2$ classifiers. The class assignment is then based on a voting between the different classifiers. This is implemented by evaluating the classifier between the i^{th} and the j^{th} class. If the classifier says $\mathbf{x}_{new}$ is in the i^{th} class this class is raised with one. The class which has received the most votes is then the winning class.

9.2 Method

the implementation of the SVM classifier, the soft margin version was used. The kernel used in this project was the Gaussian kernel. The three classes in this case is determined by the one-vs.-one setup. 250000 iterations took

9.3 Results

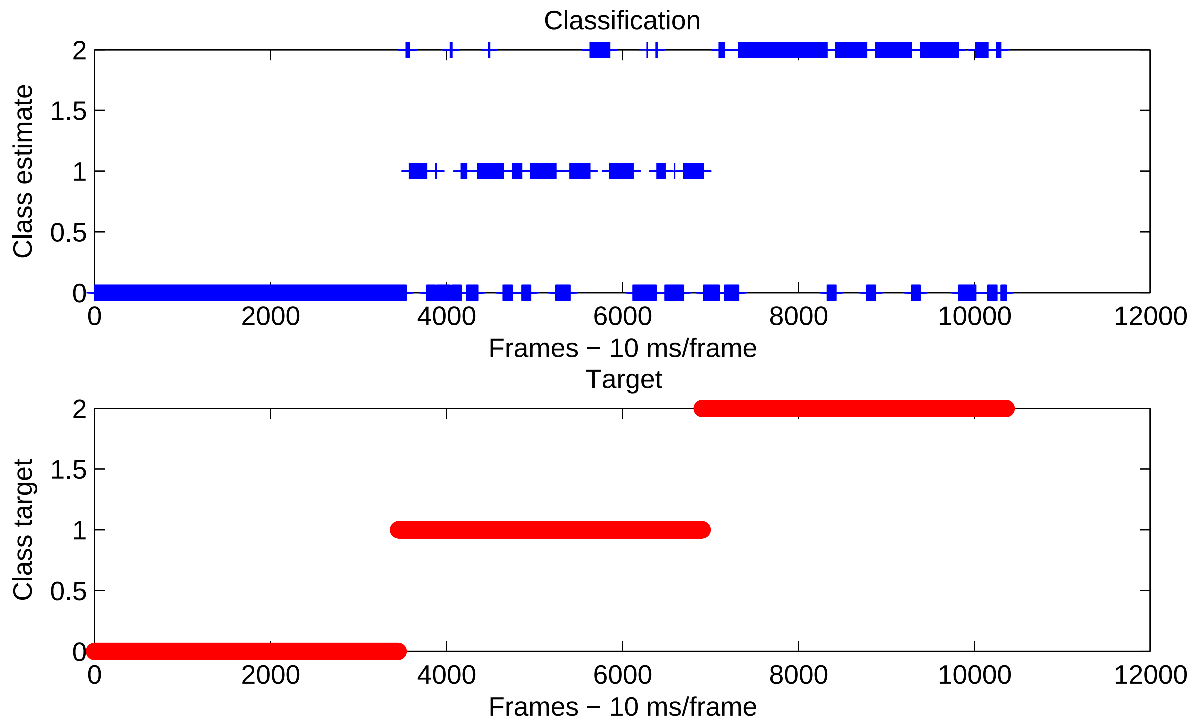

Figure 9.1: Results of using SVM classifiers and one digit spoken

	Speaker Jacob	Speaker Mose	Speaker Simon	Precision [%]
Estimate Jacob	3454.0	1408.0	1002.0	58.9
Estimate Mose	0.0	1746.0	12.0	99.3
Estimate Simon	0.0	300.0	2440.0	89.1
Sensitivity [%]	100.0	50.6	70.6	73.7

Table 9.1: Confusion matrix - 1 digit

9.4 Discussion

The SVM was applied on the dataset containing tree speakers saying one digit. The model was not applied on the other dataset because the large number of iteration it took to get a result. The other reason is that the result of the smallest dataset isn't that accurate so it was not worth applying the model on the other two datasets. The confusion matrix is shown on table 9.1 and having a overall accuracy of 73,3 %, which is okay result but in the same league as GMM or ANN. The precision of estimating speaker *Mose* is 99.3 % and the precision of estimating speaker *Jacob* is 58.9 %, this is a big difference and the reason the overall accuracy is so low. This show that the model isn't good at recognizing speakers, because it is overshooting and classify to many data points as the speaker *Jacob*. The overflow of classification for the speaker *Jacob* makes the other two classes lagging data points, so the relative high precision on the speaker *Mose* and *Simon* isn't valid.

10 Hidden Markov Models

In this project the Hidden Markov models (HMM) is not used on the data. The reason for this is that the HMM is optimized for recognizing the sequence of the sounds which combines to a word. HMM is focused on speech recognition and not speaker recognition, which is this project works with.

10.1 Theory

The HMM is developed to process sequential information in the data, in other models the features have been treated as independent and identical distribution. When classifying speech from a sequence of data recorded over time, the features tend to be strongly associate with the previous features. The associated features can be exploited by the Markov model to classify, the model uses join distributions to link the relationship between two or more successive data points $\{\mathbf{x}_1, ..., \mathbf{x}_N\}$.

$$p(\mathbf{x}_1, ..., \mathbf{x}_N) = \prod_{n=2}^{N} p(\mathbf{x}_n | \mathbf{x}_1, ..., \mathbf{x}_{n-1}) \tag{10.1}$$

The right hand-side describes the probability of a transition from one data point to the next.

If the data is noisy, an addition to Markov model can be used, which is called Hidden Markov model. The HMM uses discrete hidden states $\{\mathbf{z}_1, ..., \mathbf{z}_N\}$. The jointed distributions in the HMM can be formulated so

$$p(\mathbf{X}, \mathbf{Z} | \Theta) = p(\mathbf{z}_1 | \pi) \left(\prod_{n=2}^{N} p(\mathbf{z}_n | \mathbf{z}_{n-1}, \mathbf{A}) \right) \prod_{m=1}^{N} p(\mathbf{x}_m | \mathbf{z}_m, \phi) \tag{10.2}$$

Where $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, $\mathbf{Z} = \{\mathbf{z}_1, ..., \mathbf{z}_N\}$ and $\Theta = \{\pi, \mathbf{A}, \phi\}$ is the models parameters. The parameter π, is the initial parameter and $\mathbf{A}$ is the transition parameter, which holds the probabilities for transition between each hidden state. The parameter ϕ, is the emission parameter.

Given a set of data points the goal is to determine the model parameters in the HMM. This can be done by applying an estimation maximization algorithm on the following equation

$$Q(\Theta, \Theta^{\texttt{old}}) = \sum_{\mathbf{z}} p(\mathbf{X}, \mathbf{Z} | \Theta^{\texttt{old}}) \ln p(\mathbf{X}, \mathbf{Z} | \Theta) \tag{10.3}$$

Reference Document

[1] J. H. H. Christoffer Mose, Simon L. Madsen, “Non linear signal processing in speech recognition.”.

[2] S. P. A. E. Rosenberg, F. Bimbot, *Overview of Speaker Recognition.*

[3] M. Brookes, *VOICEBOX: Speech Processing Toolbox for MATLAB.* Department of Electrical & Electronic Engineering, Imperial College, Exhibition Road, London SW7 2BT, UK.

[4] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer, 1 ed., 2007.

[5] NETLABtoolbox, *available at: www.aston.ac.uk... /eas/research/groups/ncrg/resources/netlab/.*

[6] provided by Peter Ahrendt, “Svm notes.pdf.”.