TINONS Mini-project

Speaker Recognition

31. May 2012

Kim Bjerge (20097553)

Bjarke Møholt (20041658)

Contents

Introduction

This project encompasses implementation of different pattern classification methods for speaker recognition based on the theory presented in the course “Nonlinear signal processing and pattern recognition” (TINONS) at Aarhus University, School of Engineering. Speaker recognition systems can be characterised as *text-dependent* or *text-independent*. The methods we are aiming to develop are text-independent, meaning the system can identify the speaker regardless of what is being said.

Most of us are aware of the fact that voices of different individuals do not sound alike. This important property of speech being speaker dependent is what enables us to recognize a friend over a telephone. Speech is usable for identification because it is a product of the speaker’s individual anatomy and linguistic background. In more specific the speech signal produced by a given individual is affected by both the organic characteristics of the speaker (in terms of vocal tract geometry) and learned differences due to dialects. To consider the above concept as a basic we have tried to study different algorithms presented in the course TINONS.

Pattern recognition methods are divided into two sub-classes; supervised and unsupervised methods. For supervised methods, we need a set of pre-classified data that can be assumed to be representative for our application. For unsupervised methods, we have no prior knowledge of our data. The typical strategy for pattern recognition methods is to start by extracting features from the data set. This can be done in many ways and will often result in a high-dimension feature space. To reduce the complexity of this, dimensionality reduction methods can be applied in a clever way so that information critical for the classification remains. When this is done, supervised methods will try to group data of any single class by drawing a set of boundaries, while unsupervised methods try to group data by drawing boundaries between clusters found in the set. When the training is done, the system should be ready to recognize patterns in new data based on the decision boundaries drawn during the training, in our case the system should be able to identify the speaker in new recordings of voice.

This report covers how to make feature selection and extraction on speech signals based on the Mel-Frequency Cepstrum followed by a description of methods to reduce the feature dimensions. Feature reduction can be done by Principal Component Analysis (PCA) or by Multi-Discriminant Analysis (MDA). In the PCA method focus is to find a projection of the feature space that best represent the data in a least-square sense. The MDA method focus is to find a projection that best separates more classes from each other.

In the following chapters are described different discriminative and generative models that we have used in our work for classification of speech signals. Linear Classification and Artificial Neural Networks (ANN) are both in the category of discriminative models. Hyperplane decision boundaries as defined in linear classification are surprisingly good on a range of real-world problems. For more demanding application the approach of ANN or multilayer Perceptrons (MLP) can provide a better solution to an arbitrary classification problem. Here we seek a way to learn the nonlinearity of the problem at the same time as the linear discriminant. The project finally explores generative classification models by using a probabilistic approach. Here the Bayesian decision theory and the general multivariate Gaussian distribution are introduced. The maximum-likelihood estimation is presented which is the fundament for finding an optimal solution in the Gaussian Mixture Model (GMM). GMM is in this project used to see if it is possible to use an unsupervised learning method in finding a Gaussian mixture for two different speakers.

The theory behind these algorithms will be presented and validated by implementation of the different methods in MATLAB. In this work we will especially have focus on text-independent Speaker Recognition systems where we have studied recordings from two different speakers. Our work is based on a limited number of recordings used for training and validation of the different pattern classification algorithms presented in this report.

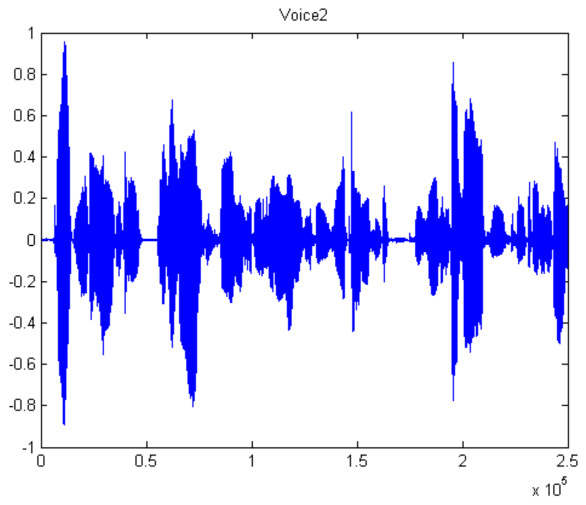


Figure 1 - Recording from speaker reading a sentence

The overall goal for this work is to demonstrate the learning objectives of the course TINONS as listed below:

- *Explain* basic terminology such as supervised/unsupervised learning, likelihood, the bias-variance relation and discriminative/generative models.
- *Compare*, *relate* and *analyze* different methods for feature extraction and feature selection on real world signals.
- *Relate* and *compare* Nonlinear Signal Processing to previously learned material such as linear FIR/IIR digital filters and adaptive filter theory.
- *Design* and *evaluate* algorithms for Linear Regression and Classification on real world signals.
- *Apply* and *explain* Artificial Neural Networks on real world signals.
- *Apply* and *explain* Gaussian Mixture Models and EM-algorithm on real world signals.
- *Apply* and *explain* Sampling Methods on real world signals.
- *Apply* and *explain* Principal Component Analyses on real world signals.
- *Apply* and *explain* Hidden Markov Models on real world signals.

Glossary

MFCC	Mel Frequency Cepstrum Coefficients
LPC	Linear Predictive Coding
FFT	Fast Fourier Transform
PCA	Principal Component Analysis
FLD	Fisher Linear Discriminant
MDA	Multiple Discriminant Analysis
LC	Linear Classifier
LMS	Least Mean Square
ANN	Artificial Neural Networks
MLP	Multi-Layer Perceptron
PBC	Probabilistic Bayesian Classifier
GMM	Gaussian Mixture Model

Theory

This chapter will describe the basic theory of the algorithms and methods used in our project. Most of the described theory is based on the book “Pattern Classification” [2].

Mel-cepstrum

A range of possibilities exist for parametrically representing the features of the speech signal for the speaker recognition, such as Linear Predictive Coding (LPC) [2] and Mel-Frequency Cepstrum Coefficients (MFCC) [1]. In this work we have chosen to use the MFCC’s coefficients that represent audio, based on perception. The MFCC is derived from the Fourier Transform of the audio clip. The basic difference between the FFT and the MFCC is that in the MFCC, the frequency bands are positioned logarithmically (on the mel scale) which approximates the human auditory system's response more closely than the linearly spaced frequency bands of FFT. This allows for better processing of data in our case for speaker recognition. The main purpose of the MFCC processor is to mimic the behavior of the human ears and brain. The MFCC process is subdivided into a number of phases or blocks as illustrated below.

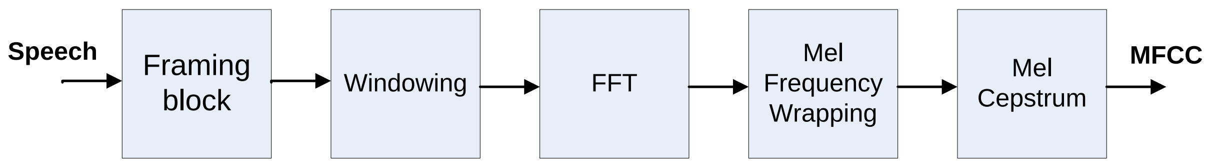


Figure 2 - Mel Cepstrum block diagram

In the frame blocking section, the speech waveform is more or less divided into frames in this work of 30 to 60 milliseconds. The windowing block minimizes the discontinuities of the signal by tapering the beginning and end of each frame to zero. The FFT block converts each frame from the time domain to the frequency domain. In the Mel frequency wrapping block, the signal is plotted against the Mel-spectrum to mimic human hearing. Studies have shown that human hearing does not follow the linear scale but rather the Mel-spectrum scale which is a linear spacing below 1000 Hz and logarithmic scaling above 1000 Hz. In the final step, the Mel-spectrum plot is converted back to the time domain by performing a Discrete Cosine Transform (DCT). The resultant matrices are referred to as the Mel-Frequency Cepstrum Coefficients. This spectrum provides a fairly simple but unique representation of the spectral properties of the voice signal which is the key for representing and recognizing the voice characteristics of the speaker. Speaker voice patterns may exhibit a substantial degree of variance: identical sentences, uttered by the same speaker but at different times, result in a similar, yet different sequence of MFCC matrices. The purpose of speaker modelling is to build a model that can cope with speaker variation in feature space and to create a fairly unique representation of the speaker's characteristics.

In order to produce a set of acoustic vectors, the original vector of sampled values is framed into overlapping blocks. Each block will contain N samples with adjacent frames being separated by M samples where M < N. The first overlap occurs at N-M samples. Since speech signals are quasi stationary between 5msec and 100msec, N will be chosen so that each block is within this length in time. In order to calculate N, the sampling rate needs to be determined. N will also be chosen to be a power of 2 in order to make use of the Fast Fourier Transform in a subsequent stage. M will be chosen to yield a minimum of 50% overlap to ensure that all sampled values are accounted for within at least two blocks. Each block will be windowed to minimize spectral distortion and discontinuities. A Hann window will be used. The Fast Fourier Transform will then be applied to each windowed block as the beginning of the Mel-Cepstral Transform. After this stage, the spectral coefficients of each block are generated. The Mel Frequency Transform will then be applied to each spectrum to convert the scale to a mel scale. The following approximate transform can be used.

$$\mathrm{Mel}(f) = 2595*\log 10\,(1 + f/700)$$

The MATLAB toolbox **voicebox** has been used to create the MFCC where we have created 12 cepstral coefficients for each sample with a window of 30 ms for each speech recordings. With a sampling rate of 44.1 kHz (fs) we have:

N = 1320 for a window size of 30 ms at 44.1 kHz

M = 660 for a minimum of 50% overlap of samples within two blocks

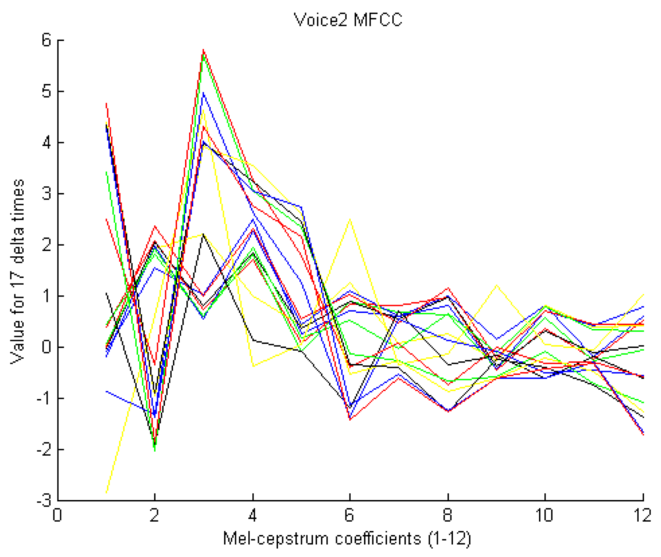


Figure 3 - Mel-Frequency Cepstrum Coefficients for 17 frame blocks

Principal Component Analysis

After extracting the features, the key point is to reduce the feature dimensionality. The principle of Principal Component Analysis (PCA) is to perform an orthogonal linear transformation projecting the data onto a new coordinate system, so that the greatest variance by any projection comes to lie on the first direction (first principal component), the second greatest variance along the second direction and so on.

We consider the problem of representing all of the vectors in a set of n-dimensional samples $x_1 \ldots x_n$ by a single vector x_0, so that the squared distances between x_k and x_0 are as small as possible. The squared error function is then

$$J_0(x_0) = \sum_{k}^{n} \|x_k - x_0\|^2$$

We want to minimize this. A 1-dimensional representation can be obtained by projecting the data onto a line that goes through the sample mean. The equation of the line is

$$x = \boldsymbol{m} + a\mathbf{e}$$

where **e** is a unit vector in the direction of the line. If we then represent x_k by $x_k = \boldsymbol{m} + a_k\mathbf{e}$, we can find an optimal set of coefficients a_k by minimizing the squared error, also known as the cost function:

$$J_1(a_1, \ldots, a_n, \mathbf{e}) = \sum_{k}^{n} \|(\boldsymbol{m} + a_k\boldsymbol{e}) - x_k\|^2$$

Remembering $|\mathbf{e}|=1$, partially differentiating with respect to a_k and setting the derivative to zero, we obtain

$$a_k = \boldsymbol{e}^t(\boldsymbol{x}_k - \boldsymbol{m})$$

We note that the best set of a_k depends on the direction **e**. Substituting the a_k we obtain

$$J_1(\mathbf{e}) = -\sum_k^n [\mathbf{e}^t(x_k - m)]^2 + \sum_k^n [x_k - m]^2 = \mathbf{e}^t\mathbf{S}\mathbf{e} + \sum_k^n [x_k - m]^2$$

$$where\ \mathbf{S} = \sum_k^n (\boldsymbol{x}_k - \boldsymbol{m})(\boldsymbol{x}_k - \boldsymbol{m})^t$$

Note that **S** only vary by a constant from the *covariance matrix* **Σ**

$$\boldsymbol{\Sigma} = \frac{1}{N}\sum_k^n (\boldsymbol{x}_k - \boldsymbol{m})(\boldsymbol{x}_k - \boldsymbol{m})^t$$

It appears obvious that the **e** that minimizes J_1 also maximizes $\mathbf{e}^t\mathbf{S}\mathbf{e}$, under the constraint that |**e**|=1. To maximize $\mathbf{e}^t\mathbf{S}\mathbf{e}$, we use the method of Lagrangian multipliers and get:

$$u = \mathbf{e}^t\mathbf{S}\mathbf{e} - \boldsymbol{\lambda}\mathbf{e}^t\mathbf{e}, \qquad \frac{\partial u}{\partial \boldsymbol{e}} = 2\mathbf{S}\mathbf{e} - 2\boldsymbol{\lambda}\mathbf{e} = \mathbf{0} \rightarrow \ \mathbf{S}\mathbf{e} = \boldsymbol{\lambda}\mathbf{e}$$

From this, we can see that a minimizer **e** is one of the eigenvectors of **S**. Furthermore, the eigenvector corresponding the largest eigenvalue also represents the direction of the largest variance in the feature space and, thus, the direction we want to project our features on to in order to maintain the highest level of information through a dimensionality reduction. The second-largest eigenvalue corresponds to the eigenvector representing the direction of the second-largest variance and so on.

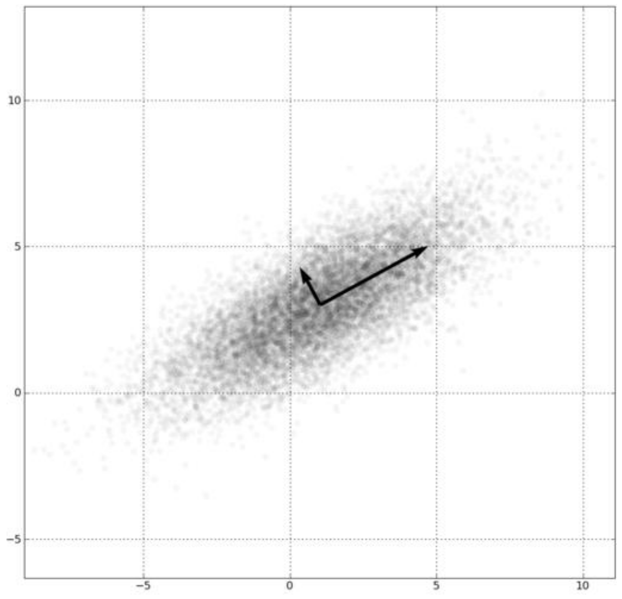


Figure 4 - First and Second Principle Component of 2D feature set

PCA enables a reduction of feature dimensionality for an unsupervised data set.

Fisher Linear Discriminant

PCA is useful for finding greatest variance and so, for representing data. However, finding the best direction for *representing* data is not necessarily the same as finding the best direction for *discriminating* data.

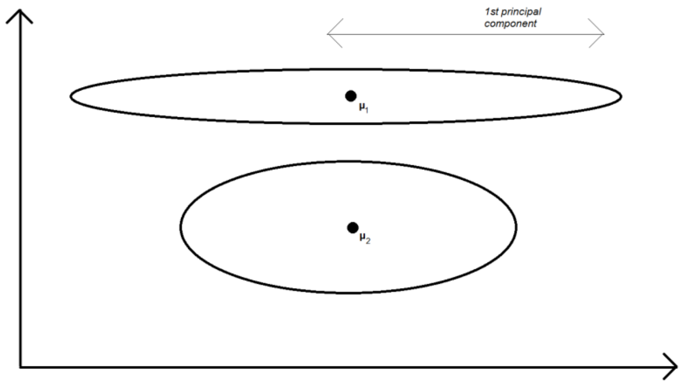


Figure 5 - PCA on two known classes

This is an example of how PCA can fail. Projecting the 2D feature space above onto the 1st principal component will make it very hard to distinguish the two classes.

For discriminating two known classes, the Fisher Linear Discriminant (FLD) method has been proposed. Assuming that we know the classes of the data we look at (supervised data), the best discrimination between the data would be a projection that 1) seeks the greatest separation between projected class means, and 2) seeks to minimize the projected variance of the classes. For discriminating more than two classes, Multiple Discriminant Analysis, which is a *c*-class generalization of the FLD, is used, so I'll start with the Fisher Linear Discriminant:

FLD begins by supposing we have a set of *n* samples $\mathbf{x}_1, \ldots, \mathbf{x}_n$. These samples are divided into the subsets D_1 and D_2. The subsets Y_1 and Y_2 are obtained from **x** by a linear projection $y = \mathbf{w}^t\mathbf{x}$, where $\mathbf{x} \in D_1$ and D_2 respectively. Having these definitions, we move on to finding the best direction of **w** for separating the data of the subsets, or, as it turns out, for getting the greatest difference in the sample mean values of the subsets: Let $\mathbf{m}_i$ be the sample mean given by

$$\boldsymbol{m}_i = \frac{1}{n_i}\sum_{x \in D_i} \boldsymbol{x}$$

then the sample mean for the projected points is given by

$$\widetilde{m}_1 = \frac{1}{n_i}\sum_{y \in Y_i} \boldsymbol{y} = \frac{1}{n_i}\sum_{x \in D_i} \boldsymbol{w}^t\boldsymbol{x} = \boldsymbol{w}^t\boldsymbol{m}_i$$

This can be seen as the sample mean projected onto **w**. From this we can derive the distance between the projected means as

$$|\widetilde{m}_1 - \widetilde{m}_2| = |\boldsymbol{w}^t(\boldsymbol{m}_1 - \boldsymbol{m}_2)|$$

We define the scatter for projected samples by

$$\tilde{s}_i{}^2 = \sum_{y \in Y_i} (y - \tilde{m}_i)^2$$

As described before, FLD employs a linear function $\mathbf{w}^t\mathbf{x}$ for which the criterion function

$$J(\mathbf{w}) = \frac{|\tilde{m}_1 - \tilde{m}_2|}{\tilde{s}_1{}^2 + \tilde{s}_2{}^2}$$

Is maximized. To obtain the **w'** maximizing *J*(**w**) we define the scatter matrices $\mathbf{S}_i$ and $\mathbf{S}_w$ by

$$\mathbf{S}_i = \sum_{\mathbf{x} \in D_i} (\mathbf{x} - \mathbf{m}_i)\,(\mathbf{x} - \mathbf{m}_i)^t$$

$$\mathbf{S}_w = \mathbf{S}_1 + \mathbf{S}_2$$

Then

$$\tilde{s}_i{}^2 = \sum_{x \in D_i} (\boldsymbol{w}^t\boldsymbol{x} - \boldsymbol{w}^t\boldsymbol{m}_i)^2 = \boldsymbol{w}^t\mathbf{S}_i\boldsymbol{w}$$

$$\tilde{s}_1{}^2 + \tilde{s}_2{}^2 = \boldsymbol{w}^t\mathbf{S}_w\boldsymbol{w}$$

Similar to this we write

$$|\tilde{m}_1 - \tilde{m}_2|^2 = (\boldsymbol{w}^t\boldsymbol{m}_1 - \boldsymbol{w}^t\boldsymbol{m}_2)^2 = \boldsymbol{w}^t\mathbf{S}_B\boldsymbol{w}$$

where

$$\mathbf{S}_B = (\boldsymbol{m}_1 - \boldsymbol{m}_2)(\boldsymbol{m}_1 - \boldsymbol{m}_2)^t.$$

Expressing the criterion function *J*(**w**) using $\mathbf{S}_W$ and $\mathbf{S}_B$ we can write

$$J(\mathbf{w}) = \frac{\boldsymbol{w}^t\mathbf{S}_B\boldsymbol{w}}{\boldsymbol{w}^t\mathbf{S}_w\boldsymbol{w}}$$

This expression is known from physics as the generalized Rayleigh quotient. Such quotients have the property that the **w** that maximizes *J* must satisfy

$$\mathbf{S}_B\boldsymbol{w} = \lambda\mathbf{S}_w\boldsymbol{w}$$

for some constant λ. It is fairly easy to turn the above equation into an eigenvalue problem and solve for **w**. $\mathbf{S}_W$ is called the within-class scatter and $\mathbf{S}_B$ the between-class scatter. The direction **w** is the optimal direction to project the data in order to separate them.

As mentioned before, Multiple Discriminant Analysis is a generalization of the Fisher Linear Determinant for a *c*-class problem, leading to *c* – 1 projection directions:

$$y_i = \mathbf{w}_i{}^t\mathbf{x}, i = 1, \ldots, c - 1.$$

Moving from vector to matrix form, instead we write

$$\boldsymbol{y} = \boldsymbol{W}^t\boldsymbol{x}$$

The projected samples can be described by their own means and scatter matrices:

$$\tilde{\mathbf{m}} = \frac{1}{n}\sum_{i}^{c} n_i\tilde{\mathbf{m}}_i$$

$$\tilde{\mathbf{S}}_W = \sum_{i}^{c}\sum_{y\in Y_i}(\mathbf{y}-\tilde{\mathbf{m}}_i)(\mathbf{y}-\tilde{\mathbf{m}}_i)^t$$

$$\tilde{\mathbf{S}}_B = \sum_{i}^{c} n_i(\mathbf{x}-\tilde{\mathbf{m}}_i)(\mathbf{x}-\tilde{\mathbf{m}}_i)^t$$

From here, it is easy to see that

$$\tilde{\mathbf{S}}_W = \boldsymbol{W}^t\mathbf{S}_w\boldsymbol{W}$$

$$\tilde{\mathbf{S}}_B = \boldsymbol{W}^t\mathbf{S}_B\boldsymbol{W}$$

The generalized criterion function *J*(**W**) can be written in terms of $\tilde{\mathbf{S}}_W$ and $\tilde{\mathbf{S}}_B$ as

$$J(\boldsymbol{W}) = \frac{\det(\tilde{\mathbf{S}}_B)}{\det(\tilde{\mathbf{S}}_W)} = \frac{\det(\boldsymbol{W}^t\mathbf{S}_B\boldsymbol{W})}{\det(\boldsymbol{W}^t\mathbf{S}_w\boldsymbol{W})}$$

Finding a matrix **W** that maximizes *J* can be done by considering the Rayleigh quotient property

$$\mathbf{S}_B\boldsymbol{w}_i = \lambda_i\mathbf{S}_w\boldsymbol{w}_i$$

Finding the vectors $\mathbf{w}_i$ gives us the best directions to project our data in order to separate the different classes. The maximum number of directions is *c-1* for a *c*-class problem.

Linear Classifier

To recognize known classes it is necessary to distinguish between them. Classifiers are mathematical tools designed for this purpose. A linear discriminant function can be written as

$$y(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x} = \boldsymbol{w}^t\begin{bmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{bmatrix}$$

Where **w** is the *weight vector*. The last element in **w** makes for the *bias*.

To train a classifier in a supervised set of samples, we define the *excitation functions* $t_i(\boldsymbol{x})$ as

$$t_i(\boldsymbol{x}) = 1\ if\ \boldsymbol{x} \in c_i, 0\ otherwise$$

We also define the *excitation matrix* **T** as

$$T = \begin{matrix} t_1(\boldsymbol{x_1}) & \dots & t_k(\boldsymbol{x_1}) \\ \vdots & \ddots & \vdots \\ t_1(\boldsymbol{x_n}) & \dots & t_k(\boldsymbol{x_n}) \end{matrix}$$

For each y_i we want to find a set of coefficients $\boldsymbol{w}$ that gives us the best linear approximation, by considering the linear equation $y_m(\boldsymbol{x}) = \boldsymbol{w}_m{}^T\boldsymbol{x}$

There are numerous methods for this problem; we have used linear least-squares regression:

We define the cost function

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{i}^{n}(y_m(\boldsymbol{x}_i) - t_m(\boldsymbol{x}_i))^2 = \frac{1}{2}\sum_{i}^{n}(\boldsymbol{w}_m{}^T\boldsymbol{x}_i - t_m(\boldsymbol{x}_i))^2$$

for each class c_m. Next we minimize this cost by differentiating for **w** and solving for 0 and get

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{T}$$

Doing so for each class, we are now able to find the decision boundary between any two classes by finding the intersection of the linear approximations. For example, we want to separate c_1 and c_2; we find the linear approximations y_1 and y_2 using the method described above and can now describe the decision boundary by $y_1 = y_2$.

Note that this method has its weaknesses: Outliers have a comparably large weight due to the squared errors regression used. Also, samples should be weighted by $\frac{N}{N_i}$ so that any uneven distribution of training samples among the classes is evened out.

Artificial Neural Networks

Multilayer Perceptron (MLP) or Artificial Neural Networks (ANN) implements linear discriminants like the linear regression classifiers, but in a space where the inputs are mapped nonlinearly. MLP are fairly simple algorithms where the form of the nonlinearity can be learned from training data and applied to a number of real-world applications. The most popular method for training a MLP network is the *backpropagation* algorithm that is a natural extension to the LMS algorithm. In this chapter we will only go in detail with the MLP network architecture and how to use the training algorithm.

We are using a three-layer neural network which consists of an input layer, a hidden layer and output layer. The layers are interconnected with modifiable weights. Each hidden unit computes the weighted sum of its inputs to form a scalar net activation. The hidden layer and the output layer emit an output that is a nonlinear function of its activation. For classification each feature dimension are assigned to an input and each class to an output of the MLP network. For speaker recognition the inputs would be the Mel-spectrum coefficients and the output the different speakers to classify.

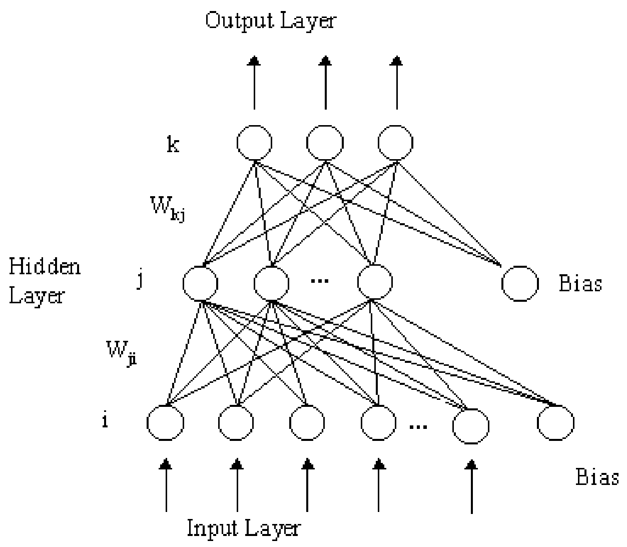


Figure 6 - Multilayer Neural Network with three-layers

The output discriminant functions can be expressed as

$$Y_k(x) = f\,\left(\sum_j w_{kj}\, f\left(\sum_i w_{ji} x_i \;+\; w_{j0}\right) + w_{k0}\,\right)$$

Each output unit computes its net activation based on the hidden unit signals. Different types of activation functions can be used. The activation functions are nonlinear to ensure that points close to the discriminate line has the biggest influence on the classification. In the following we will describe the logistic sigmoid, tanh and softmax activation functions. The sigmoid is smooth, differentiable, nonlinear and saturating. The softmax function is similar to a probability estimate with values between 0 and 1. The equations for each activation function are described below.

The *Logistic Sigmoid* activation function ensures values between 0-1 and is good for two classes

$$y(\boldsymbol{x}) = \frac{1}{1 + \; e^{-\,\mathrm{w}^t\, x\, + w_0}}$$

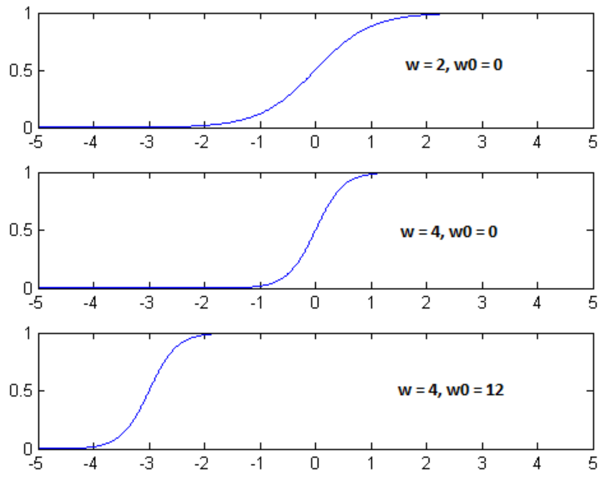


Figure 7 - Logistic activation function with variation of w (slope) and w0 (offset)

The *tanh* activation function

$$y(\boldsymbol{x}) = \tanh(\boldsymbol{w}^t\boldsymbol{x}) = 2\frac{1}{1+\ e^{-\mathrm{w}^t x + w_0}} - 1$$

The generalized *Softmax* activation function for output y_j where $\sum_j y_j(x) = 1$ and $y_i(x) > 0$, and good for more than two classes

$$y_j(x) = \frac{e^{\,\boldsymbol{w}_j^t\boldsymbol{x}}}{\sum_j e^{\,\boldsymbol{w}_j^t\boldsymbol{x}}}$$

The steps in using and training a MLP network for classification are:

1. Choose the MLP network where the number of inputs is equal to the dimension or number of features. The number of outputs reflects the number of classes to determine. An appropriate activation function is selected.
2. A training set is selected and must be proportional to the number of chosen hidden units. The network is trained using e.g. the backpropagation algorithm.
3. The trained MLP network is validated on a test set.

We define a cost function that needs to be minimized in order to find the best MLP network.

$$J(w) = \sum_i (y(x_i) - \ t_i)^2 + \propto \sum_i |w_i|^2$$

The criterion function is defined as the sum of square errors of the training error to where a regularization term is added. The regularization term adds a value to the training error where we take into account the complexity of the network. The parameter alpha ($\propto$) is adjusted to impose the regularization more or less strongly. The solution for two classes we get the *cross-entropy* cost function.

$$J(w) = \sum_i \left(t_i^1 ln y_1(x_i) + t_i^2 ln y_2(x_i) \right)$$

Training is very time consuming and it is difficult to automate since the variables are mutually dependent for an optimal training. Below are listed the variables that is dependent on an optimal training:

1. Size of the training set
2. Number of hidden units (Many hidden units requires more training data)
3. Value of the alpha valued used in the regularization term
4. Initial values of the weights
5. The Bayes error

The MLP network can never be better than the Bayes error which means it will not be better than the overlap of the class distributions. Like illustrated in Figure 8 where we have two classes C1 and C2 with a Gaussian distribution and a region of overlap that defines the minimal theoretical error.

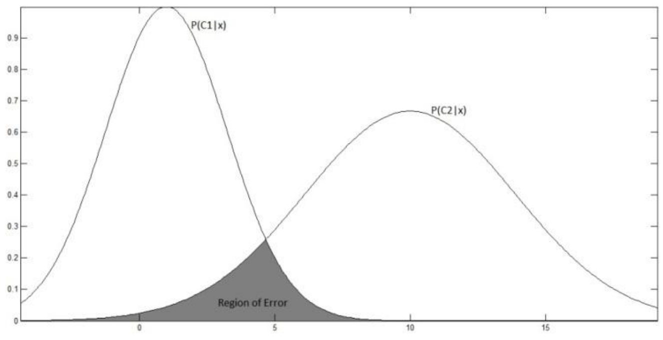


Figure 8 - Bayes error for overlapping densities

The error is typical high before the training has begun. Through learning the error becomes lower, as shown in the learning curve (Figure 9). The training error reaches an asymptotic value which depends on the Bayes error, amount of training data and the number of weights in the network (hidden units). When to stop training will depend on a validation set. We will use the validation set as stopping criterion when the minimum gradient descent is reached. The optimal way would be to use cross-validation by using different blocks of random samples for the training and test set.

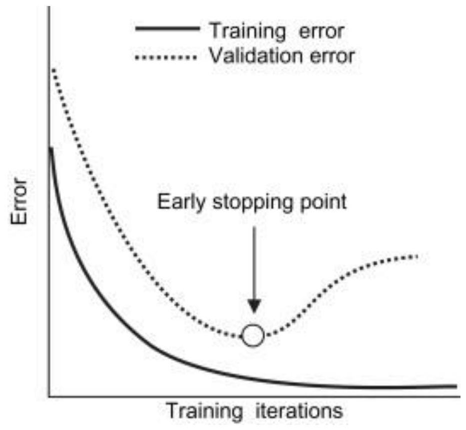
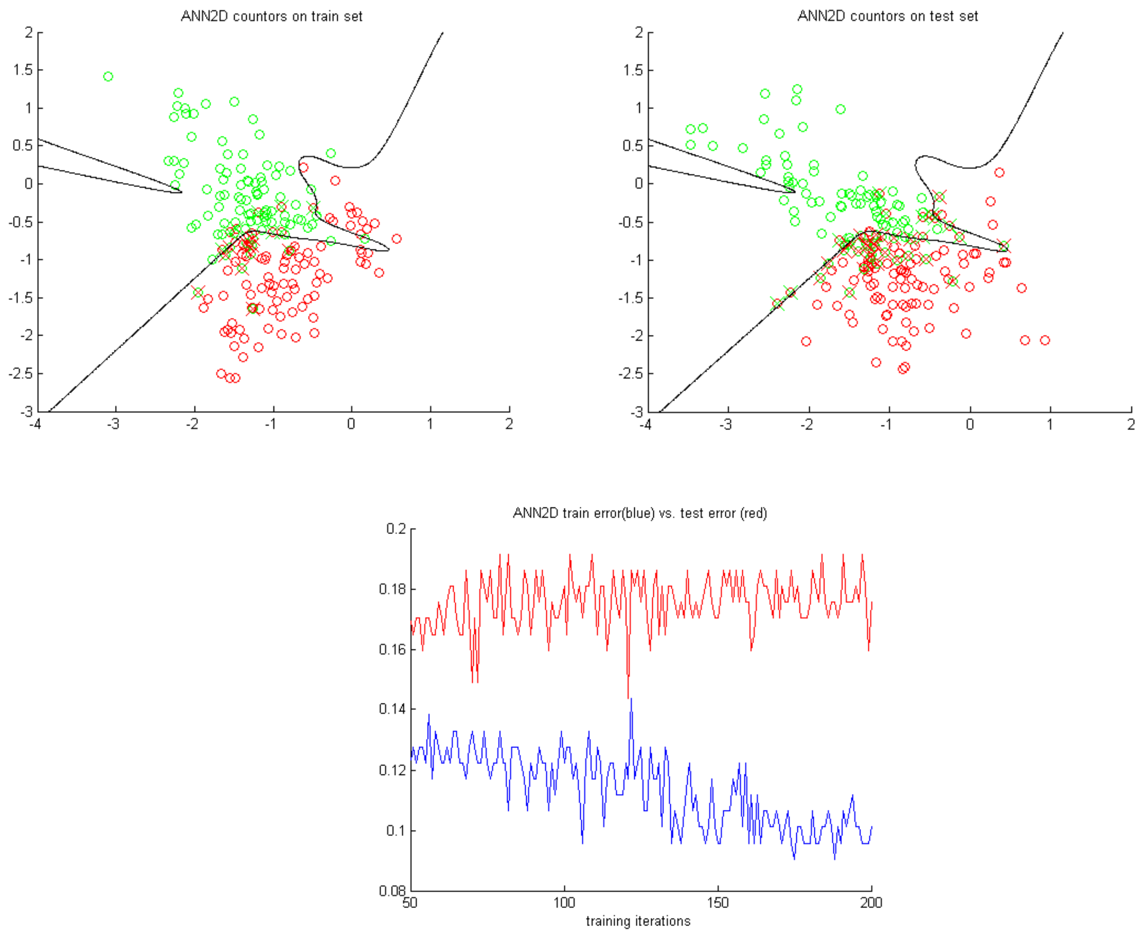


Figure 9 - Learning and validation curves

The problem we get is overfitting when selecting too many hidden units with too many training iterations or improper adjustment of the regularization term. In the example illustrated below we will get an error of 0.10 on the training data but on the test data we have an error of 0.20 after 200 training iterations.



Bayesian Classifier / probabilistic classifier

Again considering a supervised set of data, we want to classify the data by soft assignment, stating a probability that a sample belongs to one or the other class, instead of the previously described hard assignment, where a hard decision boundary assigns a 100% association to a single class.

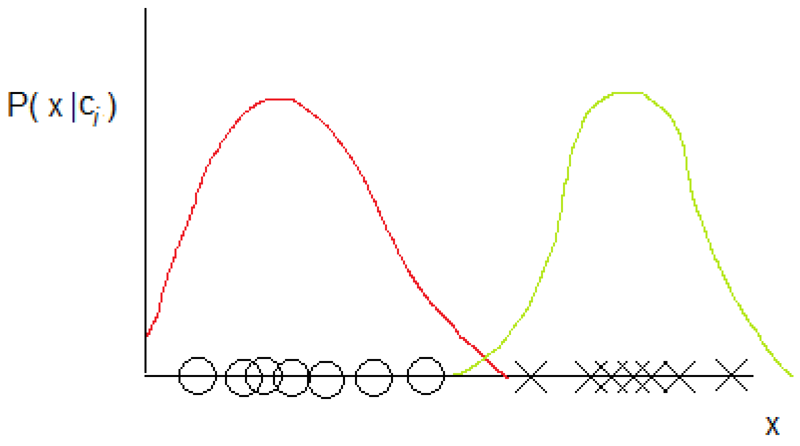


Figure 10 - probability density of two classes

Fig. 3 shows a set of 1-D samples belonging to two classes, and their distribution functions **P(x|c$_k$)**. Knowing that $\boldsymbol{P(c_i)}$ can be found by $\frac{N_i}{N}$ assuming that the distribution of samples in the training set is representative, we can calculate the probabilities of a given sample belonging to a class **P(c$_k$|x)** by utilizing Bayes' Theorem:

$$\boldsymbol{P(c_k|x)} = \frac{\boldsymbol{P(x|c_k)P(c_k)}}{\boldsymbol{P(x)}} = \frac{\boldsymbol{P(x|c_k)P(c_k)}}{\sum_i \boldsymbol{P(x|c_i)P(c_i)}}$$

For example, if we know the probability densities of two classes and we have an unclassified sample **x**$_{\text{new}}$, we could calculate the probability of it belonging to class 1 by

$$\boldsymbol{P(c_1|x_{new})} = \frac{\boldsymbol{P(x_{new}|c_1)P(c_1)}}{\sum_i \boldsymbol{P(x_{new}|c_i)P(c_i)}} = \frac{\boldsymbol{P(x_{new}|c_1)P(c_1)}}{\boldsymbol{P(x_{new}|c_1)P(c_1)} + \boldsymbol{P(x_{new}|c_2)P(c_2)}}$$

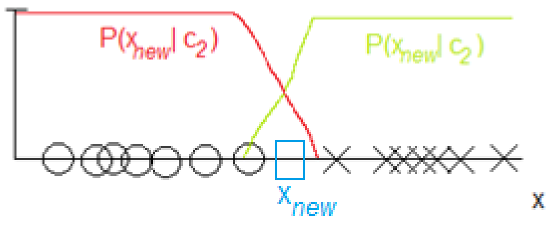


Figure 11 - probability of a sample belonging to one or the other class

The above sketch also shows a decision boundary, being where $\boldsymbol{P(x_{new}|c_1) = P(x_{new}|c_2)}$. Note that the probability of **x** belonging to another class than the class indicated by the decision boundary is not 0, which means that it is to be expected that a portion of the samples will be incorrectly classified. We denote this as the Bayes error.

Gaussian Mixture Model

Unsupervised methods can be used to find patterns in data without training. Collecting and labeling a large set of sample patterns can be costly. In unsupervised learning we achieve to find methods that can be used to decide on patterns for features in classification. We will train with a large unlabeled set of data, but we still have to use supervision to label the groupings found in the data. In our project we could use such an approach in recording speech from different speakers and to use unsupervised training in looking for groups/clusters of patterns that matches the individual speaker's identity.

A popular approximation method is the k-Means algorithm that could be used for unsupervised classification finding clusters of patterns. The method is more simple that the Gaussian Mixture Model (GMM) and could be used as a way to find means for every cluster in the training data set. The method is used to compute and accelerate the convergence of finding clustering patterns in the sample data by means of the k-Means algorithm as summarized below

1. Choose a value of *k* the number of clusters, given the number of samples *n*
2. Initiate the mean values μ_i for each cluster by choosing random samples
3. Assign sample points x_n to each mean cluster using the minimum Euclidian distance
4. Calculate new mean values: $\mu_i^{(j+1)} = \frac{\sum_n r_{ni} x_n}{\sum_n r_{ni}}$, where the *hard responsibility* $r_{ni}$is 1 if x_n belongs to cluster *i*, 0 otherwise.
5. Iterate point 3+4 until means variance between iterations is below a threshold or that a defined cost function based on the squared Euclidian distance to the mean is small. The cost function is

$$J = \sum_n \sum_i r_{ni} \, |x_n - \mu_i|^2$$

If we assume that the complete probability structure for the problem can be described by a normal distribution of each cluster, we could use GMM. We have to find the unknown parameters of the probability distribution and the number of components/clusters (*k*). Assuming that the distribution for each component is Gaussian we need to find the unknown variance and mean parameters: $\theta_i = (\sigma_i, \mu_i)$

The probability density function for the samples is then given by

$$p(\boldsymbol{x}|\theta) = \sum_{i=1}^{k} P(w_i) p(\boldsymbol{x}|w_i, \theta_i)$$

where $\theta = (\theta_1, \theta_2, \dots, \theta_k)$ and *k* are the unknowns. The conditional densities $p(x|w_i, \theta_i)$ are called *component densities*, and the prior probabilities $P(w_i)$ are called the *mixing parameters*. For the Gaussian Mixture Model (GMM) the conditional densities are the multivariate normal distribution.

$$p(\boldsymbol{x}|w_i, \theta_i) = \frac{1}{(2\pi)^{d/2}\sqrt{|\boldsymbol{\Sigma}|}} \, e^{\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^t \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)\right]}$$

where $\sum$ is the *covariance matrix* and *d* is the feature dimensions of the sample. We can choose 3 different types of the covariance matrix: isotropic/spherical where $\Sigma = \sigma^2 I$, diagonal where $\Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}$ and full where $\Sigma = \frac{1}{N}\sum_i (\boldsymbol{x_i} - \boldsymbol{\mu})(\boldsymbol{x_i} - \boldsymbol{\mu})^T$.

To find the optimal solution to GMM we will use the maximum-likelihood to estimate the unknown parameters which is similar to find the maximum of the log-likelihood given by

$$l(\theta) = \sum_{j=1}^{n} \ln p(x_j|\theta)$$

where we have set of *n* samples that we assume are *independent and identical distributed* (i.i.d.). From the probability density function we get

$$l(\theta) = \sum_{j=1}^{n} \ln \sum_{i=1}^{k} p(w_i)p(x_j|w_i)$$

here the mixing coefficient are called π_i, we have

$$l(\theta) = \sum_{j=1}^{n} \ln \sum_{i=1}^{k} \pi_i p(x_j|w_i)$$

where $p(x_j|w_i)$ is a normal distribution. We can write our solution formally as the argument θ that maximizes the log-likelihood

$$\hat{\theta} = \arg max\ l(\theta)$$

In finding the maximum we need to differentiate the log-likelihood according to the parameters: $\theta(\pi_i, \sigma_i, \mu_i)$ to find:

$$\nabla_\theta l = 0$$

The expectation-maximization or EM algorithm is used to iteratively estimate the likelihood for the above problem and finding the optimal parameters for the solution. We start with a guess for the parameters $\theta(\pi_i, \sigma_i, \mu_i)$. Then we used the Bayes formula to compute the probability for the samples belonging to class w_i in the E-step

$$p(w_i|\boldsymbol{x}) = \frac{P(w_i)p(\boldsymbol{x}|w_i)}{\sum_j P(w_j)p(\boldsymbol{x}|w_j)}$$

where $p(x|w_i) = N(\mu_i, \Sigma_i)$ is the normal distribution. In the M-step we compute new estimates for the means μ_i, covariance matrices Σ_i and mixing coefficients π_i.

$$N_i = \sum_{n=1}^{N} p(w_i|x_n)$$

$$\mu_i = 1/N_i \sum_{n=1}^{N} p(w_i|x_n)x_n$$

$$\Sigma_i = 1/N_i \sum_{n=1}^{N} p(w_i|x_n)(x_n - \mu_i)(x_n - \mu_i)^t$$

$$\pi_i = \frac{N_i}{N}$$

We continue to iterate between the EM-steps until the computed log-likelihood stops changing, reaches a certain steady value like 0.001. Alternatively we stop when the number of iterations exceeds a maximum. The algorithm is sensitive to estimates of the covariance matrix. When the eigenvalues becomes very small, the value of $p(x|w_i)$ goes to infinite, therefore the eigenvalues (σ) of the covariance are typical reset to 1 in this case.

Conduct of Experiments

In this project we are aiming to experiment with different methods to distinguish between different speakers. The methods we are aiming to develop are text-independent, meaning the system can identify the speaker regardless of what is being said. We have limited our experiments to focus on text-independent algorithms that would be able to identify two different male voices. We have made a number of recordings from where training and test data can be produced.

Audio recordings

The recordings are recorded using the audio/MIDI-sequencer Cubase from Steinberg. All recordings are normalized and stored in .wav files with a sampling rate of 44.1 kHz.

We recorded speech from two different voices. We made 3 scenarios:

- Repeated 'Op'
- Repeated 'Ned'
- A full sentence

"Ned" consists of the 3 phonemes n, e and ð. "Op" consists of the 2 phonemes ʌ and b. The full sentence contained a lot of different phonemes.

The thought of the scenarios was to confirm correct voice classification with very few phonemes and, as we got better at classifying, work our way towards a higher variance in phonetic content. The length of the recordings varied between 3 – 6 seconds.

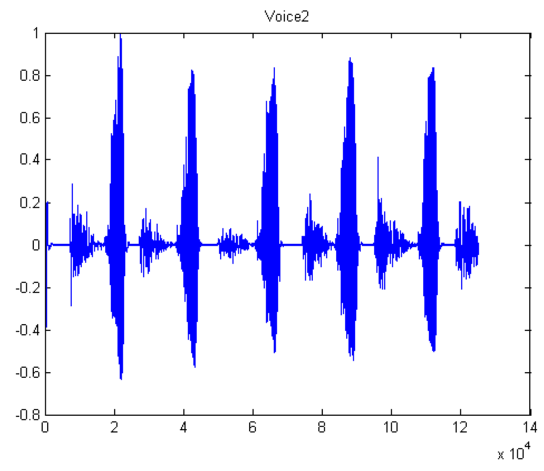
A MATLAB function is made that creates the sample features set for the recordings. With the `CreateMFCCSamples` function it is possible to specify the type of recordings from where the sample feature set should be generated. The result is a sample set for voice 1, 2 and silence, containing a feature set of 12 MFCC coefficients.

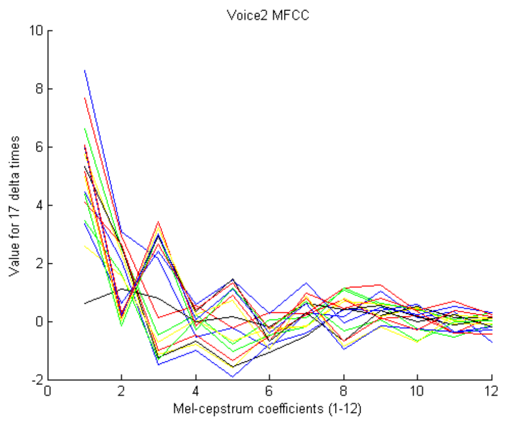
```
function [mfcc_voice1, mfcc_voice2, mfcc_silence] = CreateMFCCSamples(PlotMFCC, Pause, Start, End)
```

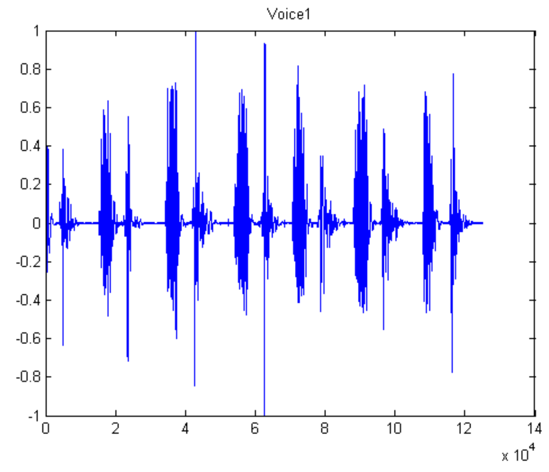
The parameters for the function specify whether to plot the audio clip and Mel-Frequency Cepstrum Coefficients. The `Start` and `End` parameters specifies the recordings that should be used in extracting the sample feature set. Specifying `Start=0` and `End=5` gives a set of samples that contains MFCC samples of all recordings from all wave files. There are 6 different recoding sets as illustrated in the table below.

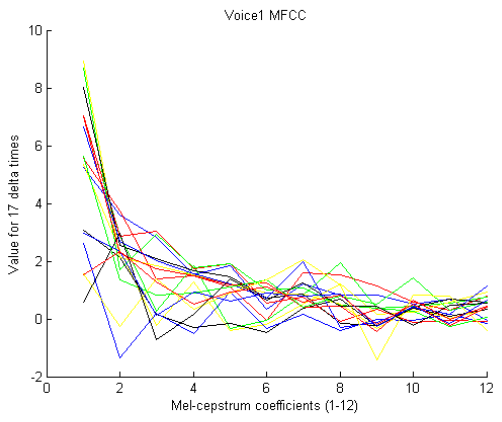
Audio file names	**Description**	**MFCC Samples**	**Start, End parameter values**
OpBjarkeC.wav, OpKimC.wav, Silence.wav	Recordings of speaker voice pronouncing "Op" (repeated)	188	0
NedBjarkeC.wav, NedKimC.wav, Silence.wav	Recordings of speaker voice pronouncing "Ned" (repeated)	188	1
Speech1_1.wav, Speech2_1wav, Silence2.wav	Reading sentence 1	377	2
Speech1_2.wav,	Reading sentence 1 (Same as above)	377	3

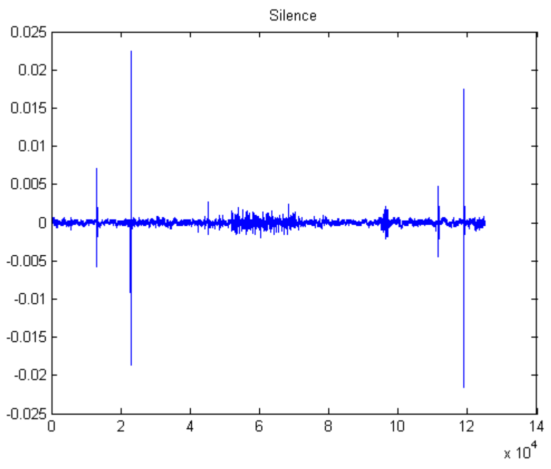
Speech2_2wav, Silence2.wav			
Speech1_A.wav, Speech2_Awav, Silence2.wav	Reading sentence A	377	4
Speech1_B.wav, Speech2_B.wav, Silence2.wav	Reading sentence B	377	5

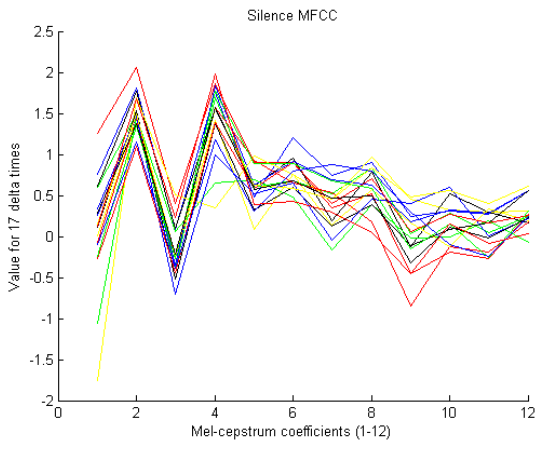












The above audio recordings visualize the speaker one and two pronouncing the word "Op" repeated a number of times. The plot of the 12 MFCC coefficients varies over 17 delta time intervals. The window size is 1320 audio samples with a step size of 660 that means the MFCC plots covers the variation over 17*660 = 11220 audio samples or approx. 250ms at the sample rate of 44.1 kHz.

MATLAB program

All experiments described in this report are combined in one MATLAB program with the purpose of exploring the different methods and algorithms for classification. The program calls functions to generate the training and test data sets, performs feature reduction and plotting the sample feature space. Finally it uses the different classification methods and algorithms as described in this report. A number of control parameters can be set to specify the execution of the program (`VoiceClassificationAllRand.m`) as listed in the MATLAB code below.

```
UsePCA_MDAFeatureReduction = 2 % 0 = none, 1 = PCA, 2 = MDA
% Classification Methods:
%                            0 = 2D, 1 = 3D,
%                            2 = ANN2D, 3 = ANN3D,
%                            4 = Bayesian decision theory,
%                            5 = GMM2D, 6 = GMM3D, 7 = GMM2DComp
UseClassificationMethodStart = 2
UseClassificationMethodEnd = 2
UseSizeTrainSet = 94
UseSizeTestSet = 94
UseRandomisation = 1

% Start, End parameters
% 0,1  Op/Ned
% 2,2  Same speech
% 2,3  Same speech twice
% 2,5  All speeches
% 0,5  All recordings
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(2, 0, 0, 0);
```

Feature reduction can be specified to use the PCA (1) or MDA (2) algorithms. The classification methods can be chosen as listed in the table below. The program iterates all possible classification methods specified by the `UseClassificationMethodStart` and `UseClassificationMethodEnd` parameters.

Classification method	**Description**
0	Performs linear classification on a sample set with 2 dimensions and plotting results in 2D feature space
1	Performs linear classification on a sample set with 3 dimensions and plotting results in 3D feature space
2	Performs ANN/MLP classification on a sample set with 2 dimensions and plotting results in 2D feature space
3	Performs ANN/MLP classification on a sample set with 3 dimensions or higher
4	Uses the Baysian decision theory assuming that the training data has a normal distribution
5	Performs unsupervised classification using the Gaussian Mixture Model on the training set of 2 dimensions and plotting the result for probabilistic classification with test data
6	Performs unsupervised classification using the Gaussian Mixture Model on the training set of 3 dimensions or higher for probabilistic/generative classification with test data
7	Performs supervised classification using GMM for each of 2 classes with 2 dimensions. It finds a number of k-components of Gaussian mixtures that is then used for a generative classification.

The `UseRandomisation` parameter defines whether to randomize the MFCC sample set before selecting the training and test data set. The parameter specifies to randomize the results returned from the `CreateMFCCSamples` function before selecting the training and test data sets. Each classification method returns the *confusion matrix* for the train and test sets being able to calculate the correct classification percentage.

```
switch (UseClassificationMethod)
    case 0
        % 2D classification training set with 2 classes and 2 features
        [Ctrain, Ctest, W] = linear2D(V1new, V1tnew, V2new, V2tnew); % training
   case 1
        % 3D classification training set with 2 classes and 3 or more features
        [Ctrain, Ctest, W] = linear3D(V1new, V1tnew, V2new, V2tnew); % training
    case 2
        % 2D classification using Artificial Neural Networks
        [Ctrain, Ctest] = ANN2D(V1new, V1tnew, V2new, V2tnew, Snew, Stnew, 2); % 2 or 3 features
    case 3
        % 3D classification using Artificial Neural Networks
        [Ctrain, Ctest] = ANN3D(V1new, V1tnew, V2new, V2tnew, Snew, Stnew, size(subSet,2));
    case 4
        % Classification based on Bayesian decision theory
        % assuming a normal distribution of class features
        [t_est, Ctest] = gausianDiscriminant(V1new, V1tnew, V2new, V2tnew); % 2 features only
    case 5
        % 2D classification using the Expectation-Maximation (EM)
        % algorithm for Gaussian Mixture Models in 2 dimensions
        % A training is performed for each class V1, V2 and silence
        % finding a Gaussian mixture for each model
        [Ctrain, Ctest] = GMM2D(V1new, V1tnew, V2new, V2tnew, Snew, Stnew);
    case 6
        % 3D classification using the Expectation-Maximation (EM)
        % algorithm for Gaussian Mixture Models in 3 dimensions or more
        % A training is performed for each class V1, V2 and silence
        % finding a Gaussian mixture for each model
        [Ctrain, Ctest] = GMM3D(V1new, V1tnew, V2new, V2tnew, Snew, Stnew, size(subSet,2));
    case 7
        % 2D classification using the Expectation-Maximation (EM)
        % algorithm for Gaussian Mixture Models in 2 dimensions
        % A training is performed for each class V1, V2
        % finding Gaussian mixture components (GMM) for each class
        [Ctrain, Ctest] = GMM2DComponents(V1new, V1tnew, V2new, V2tnew, 5);
    otherwise
        % Invalid classification parameter specifier
end
```

Results

In this section we will present the results of our work with the project.

The project work has developed in parallel with the material that has been taught in the course and so, many of the results have been projected in order to be easy to visualize rather than useful for classification. As a testing metric for all our classification methods, we have used the Confusion matrix found in the **prtools** toolbox. The Confusion matrix looks like this:

$$\begin{matrix} \#\, c_1\ classified\ as\ c_1 & \#\, c_2\ classified\ as\ c_1 \\ \#\, c_1\ classified\ as\ c_2 & \#\, c_2\ classified\ as\ c_2 \end{matrix}$$

In the following, each subsection will start with a MATLAB code snippet for setting up the control parameters to perform the operations necessary to produce our results. After this, the results are shown as 2D images. For the classification methods, a confusion matrix is shown as a summary result of the classification.

Principal Component Analysis

```
UsePCA_MDAFeatureReduction = 1 % 0=none, 1=PCA, 2=MDA
UseClassificationMethodStart = 0
UseClassificationMethodEnd = 0
UseSizeTrainSet = 94
UseSizeTestSet = 94
UseRandomisation = 0
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(0, 0, 0, 0); % "Op"
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(0, 0, 1, 1); % "Ned"
```

First we perform PCA on a portion of all “Ned” samples. The following is a non-randomized set of samples:

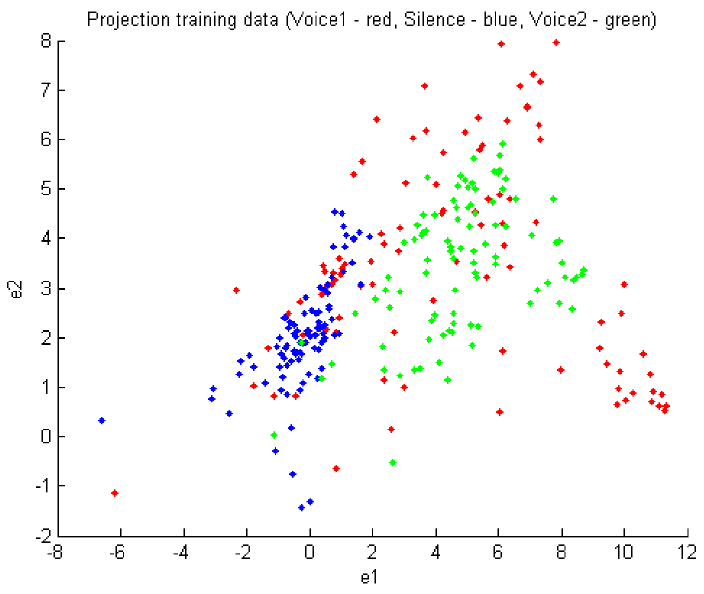


Figure 12 - PCA feature reduction in 2D for "Op" recording

Next we performed the same operation on a non-randomized set of “Ned” samples. We can almost distinguish a pattern indicating the different phonemes in the projection, as well as the speakers:

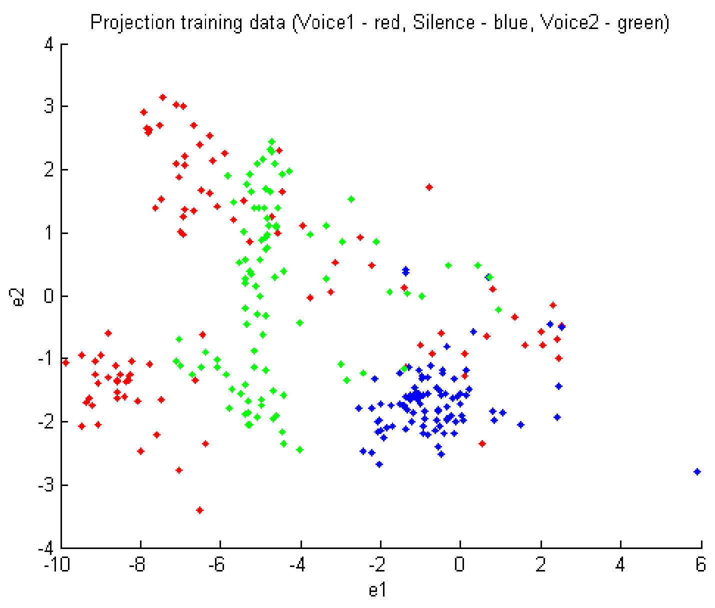


Figure 13 - PCA feature reduction in 2D for "Ned" recording

Our method assumes that our samples are independent. To check this, we also performed PCA on a random set of "Ned" samples:

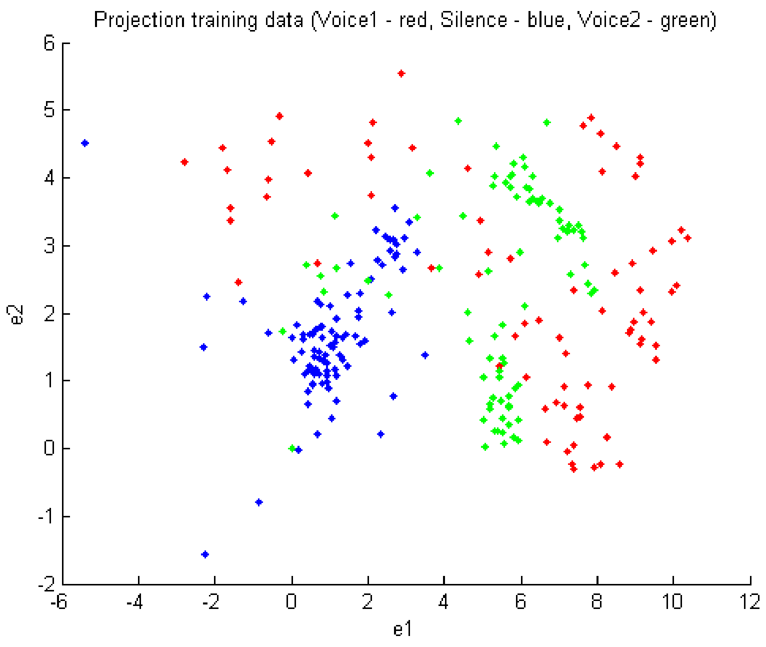


Figure 14 - PCA feature reduction of randomized "Ned" training data

This is basically the same projection as before, only the first principal component seems to be reversed compared to the non-randomized.

Multiple Discriminant Analysis

```
UsePCA_MDAFeatureReduction = 2 % 0=none, 1=PCA, 2=MDA
UseClassificationMethodStart = 0
UseClassificationMethodEnd = 0
UseSizeTrainSet = 94
UseSizeTestSet = 94
UseRandomisation = 0
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(0, 0, 0, 0); % "Ned"
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(0, 0, 1, 1); % "Op"
```

Since we were doing supervised training, we were able to use MDA for feature reduction instead of PCA. Performing MDA on 3 classes inherently limits our feature space to dimensions.

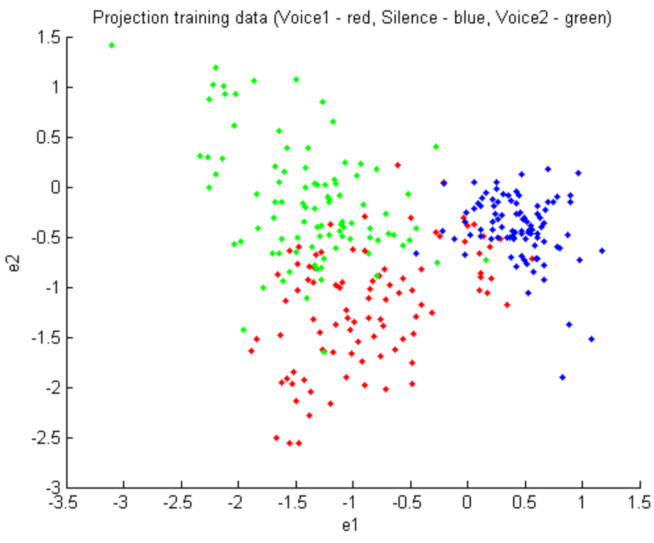


Figure 15 - MDA feature reduction in 2D for "Op" recording

We see this method is clearly better suited for grouping the samples than PCA was:

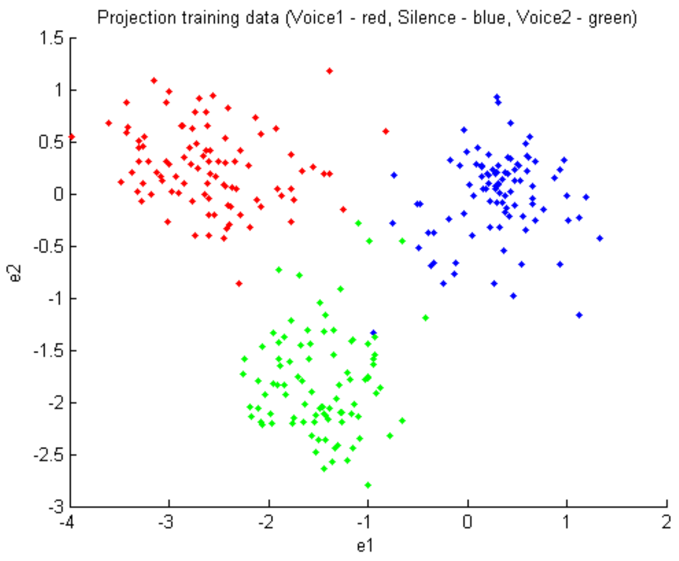


Figure 16 - MDA feature reduction in 2D for "Ned" recording

Again, checking we'd get the same results using a random set of samples, it appears that the projection has somehow 'flipped', Voice1 and Voice2 being as clearly separable but in swapped positions:

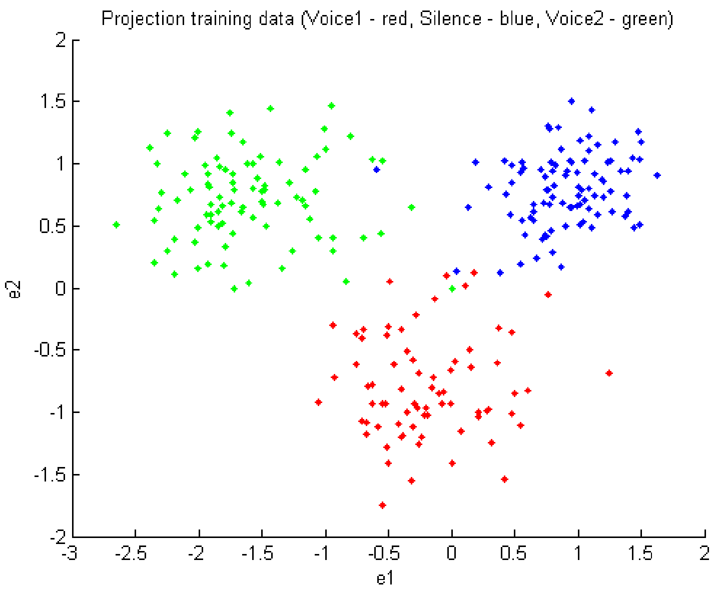


Figure 17 - MDA feature reduction in 2D for randomized "Ned" recording

Linear Classifier

```
UsePCA_MDAFeatureReduction = 2 % 0=none, 1=PCA, 2=MDA
UseClassificationMethodStart = 0
UseClassificationMethodEnd = 0
UseSizeTrainSet = 94
UseSizeTestSet = 94
UseRandomisation = 0
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(0, 0, 0, 0); % "Ned"
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(0, 0, 1, 1); % "Op"
```

LC was the first classification method we were taught and so, the first classification method we employed on our data. After our work with feature space dimensionality reduction and some initial tests with classifiers, we've decided to use MDA due to the better classification results it produces.

The first plot shows the results of the classifier training. Samples classified incorrectly are marked with an x:

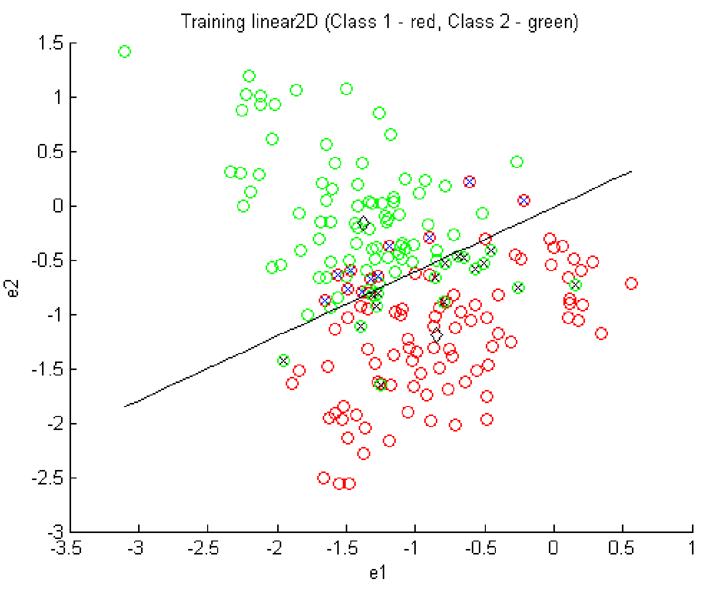


Figure 18 - Training for linear classifier on MDA feature reduction in 2D for "Op" recording

$$C_{train} = \begin{matrix} 83 & 16 \\ 11 & 78 \end{matrix} \quad err_{train} = 0.14$$

The second plot shows the results of applying the found decision boundary on a test set. No samples from the training are used in the test.

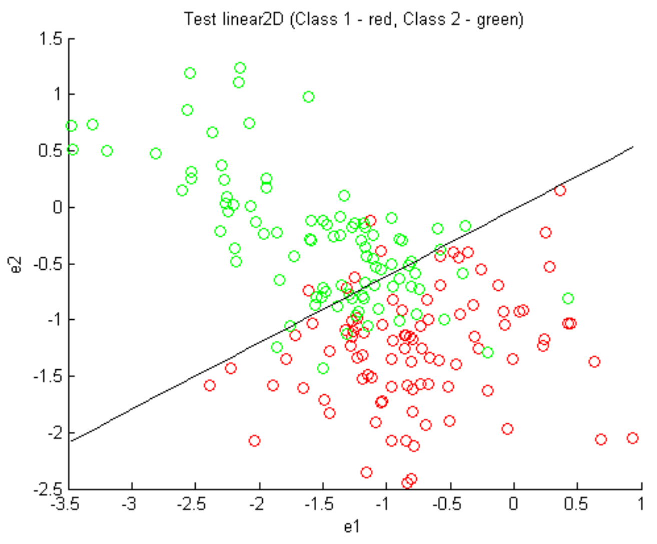


Figure 19 - Test for linear classification on MDA feature reduction in 2D for "Op" recording

$$C_{test} = \begin{matrix} 89 & 25 \\ 5 & 69 \end{matrix} \quad err_{test} = 0.16$$

As expected, the number of classification errors is a little higher in the test than in the training.

We performed the same as above on the “Ned” samples:

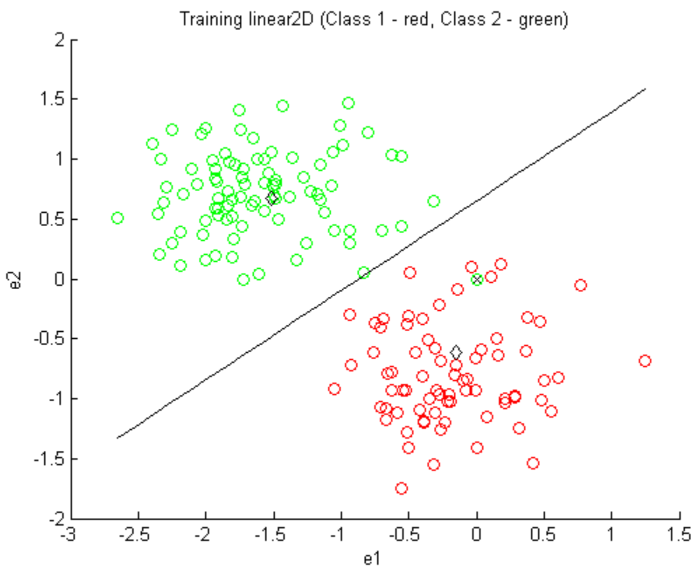


Figure 20 - Training for linear classification on MDA feature reduction in 2D for randomized "Ned" recording

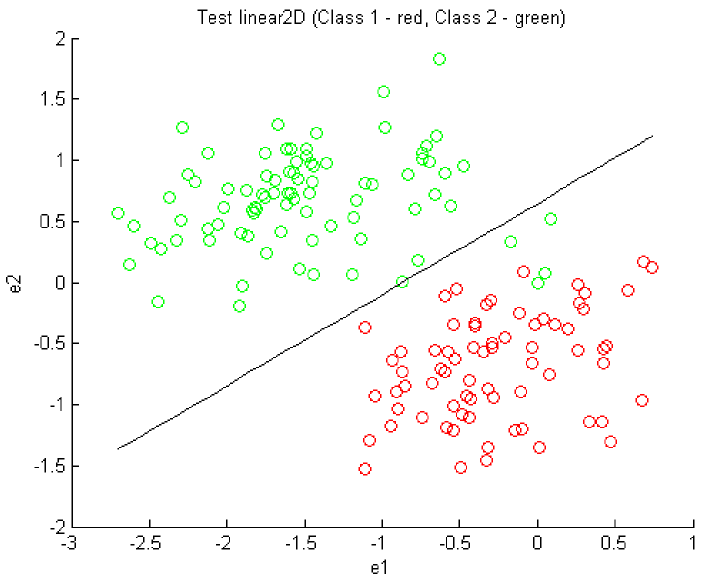


Figure 21 - Test for linear classification on MDA feature reduction in 2D for randomized "Ned" recording

$$C_{train} = \begin{matrix} 94 & 6 \\ 0 & 88 \end{matrix} \quad err_{train} = 0.03$$

$$C_{test} = \begin{matrix} 94 & 13 \\ 0 & 81 \end{matrix} \quad err_{test} = 0.07$$

Artificial Neural Networks

```
UsePCA_MDAFeatureReduction = 2 % 0=none, 1=PCA, 2=MDA
UseClassificationMethodStart = 2
UseClassificationMethodEnd = 2
UseSizeTrainSet = 94
UseSizeTestSet = 94
UseRandomisation = 0
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(0, 0, 0, 0); % "Op"

ANN2D.m – parameters

outputfunc = 'logistic';
nout = 2;                    % Number of outputs.

% Parameters to vary
nhidden = 8;                 % Number of hidden units.
alpha = 0.001;               % Coefficient of weight-decay prior (regularization).
```

This chapter presents the results of using multilayer neural networks for discriminative classification. We have chosen to use MDA feature reduction in achieving a feature dimension of 2 using recordings from voice 1, 2 and silence as illustrated below. We have selected 8 hidden units for the ANN 2D network. A logistic sigmoid output activation function is selected since we are using 2 output classes (Voice 1 and 2) for training. The coefficient alpha is chosen to 0.001 ensuring that a moderate impose of the regularization term. The parameters were found by a trial-process, varying the terms slightly and evaluating the resulting classification error.

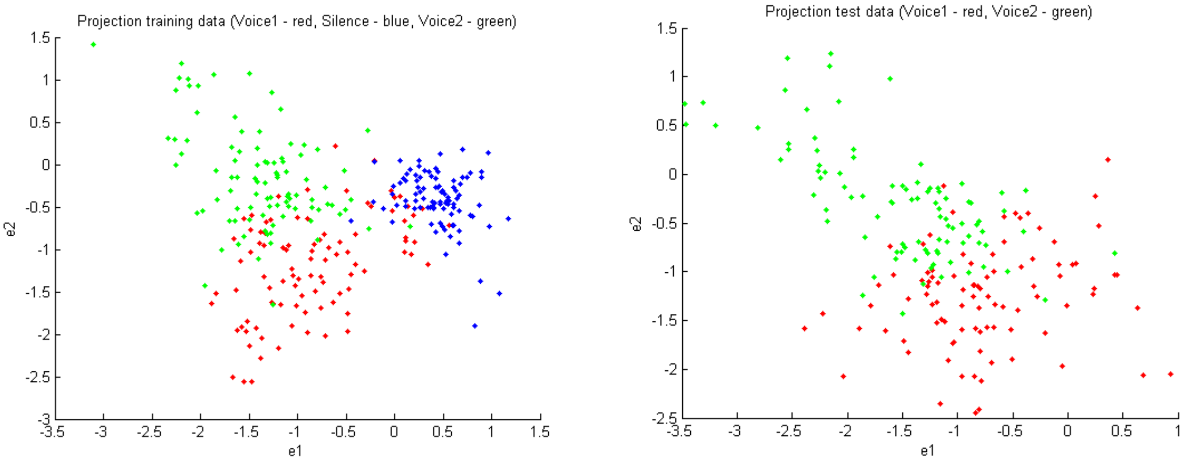


Figure 22 - Training and test set in 2D for "Op" recording

Figure 23 shows the discriminating contour that separates the features of voice 1 from voice 2. We see that the training achieves a better result than on the testing data set:

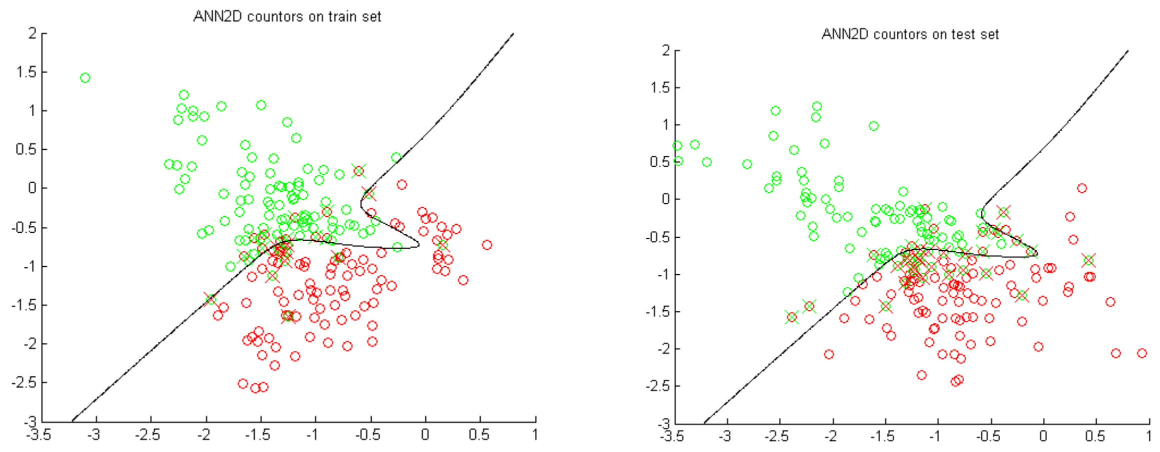


Figure 23 - ANN 2D classification in 2D for "Op" recording

Confusion matrix for the training and testing data set from the “Op” recordings.

$$C_{train} = \begin{matrix} 83 & 11 \\ 13 & 81 \end{matrix} \quad err_{train} = 0.13$$

$$C_{test} = \begin{matrix} 83 & 11 \\ 20 & 74 \end{matrix} \quad err_{test} = 0.16$$

```
UsePCA_MDAFeatureReduction = 1 % 0=none, 1=PCA, 2=MDA
UseClassificationMethodStart = 3
UseClassificationMethodEnd = 3
UseSizeTrainSet = 94
UseSizeTestSet = 94
UseRandomisation = 0
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(0, 0, 0, 0); % “Op”

ANN3D.m - parameters

outputfunc = 'softmax';
nout = 3;                   % Number of outputs.

% Parameters to vary
nhidden = 12;               % Number of hidden units.
alpha = 0.001;              % Coefficient of weight-decay prior.
```

In the next experiment we have used Multilayer neural networks with three classes. We are using PCA features reduction to reduce the 12 MFCC to 6 features. By plotting the eigenvalues (**Figure 24**) we see that the first four eigenvectors are the most significant, representing more than 95% of the variance.

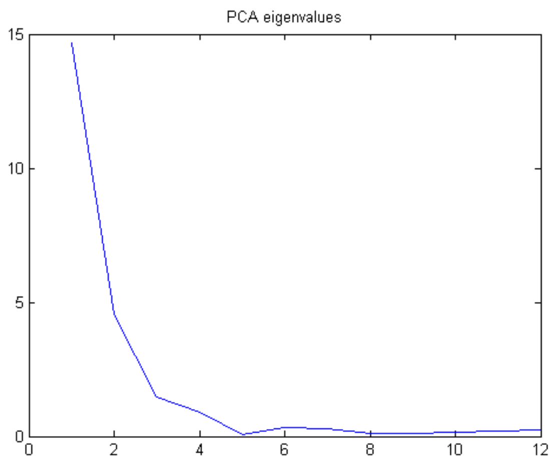


Figure 24 - Eigenvalues from the Principal Component Analysis

We have selected 6 features in trying to improve the results using ANN for 6 dimensions in classification of 3 output classes. The softmax output activation function is used since we now have more than 2 classes. We have selected 12 hidden units for the ANN 6D network.

For visualization purposes, we have omitted principal components 4-6 in the following plot:

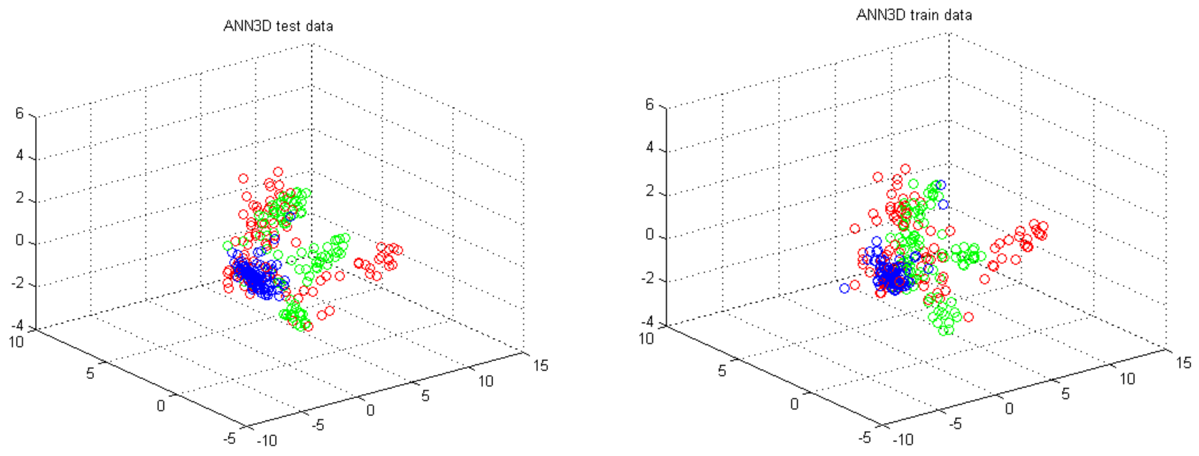


Figure 25 - Training and test set in 3D for "Op" recording

$$Ctrain = \begin{matrix} 90 & 2 & 2 \\ 4 & 89 & 1 \\ 2 & 0 & 92 \end{matrix} \quad err_{train} = 0.04$$

$$Ctest = \begin{matrix} 84 & 4 & 6 \\ 15 & 78 & 1 \\ 4 & 1 & 89 \end{matrix} \quad err_{test} = 0.11$$

Looking at the number training and test errors as function of training cycles we see in Figure 26 that an optimal number of cycles are between 40 – 60 cycles where the test errors starts to increase.

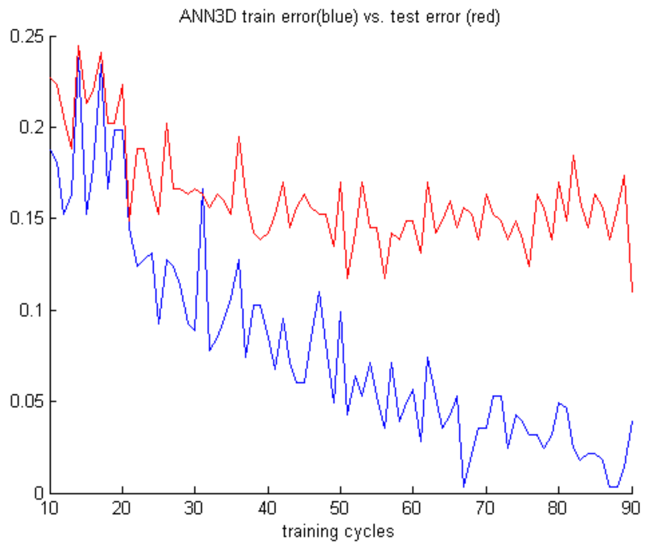


Figure 26 - Errors as functions of training cycles for training and test set

Bayesian Classifier/ probabilistic classifier

```
UsePCA_MDAFeatureReduction = 2 % 0=none, 1=PCA, 2=MDA
UseClassificationMethodStart = 4
UseClassificationMethodEnd = 4
UseSizeTrainSet = 94
UseSizeTestSet = 94
UseRandomisation = 0
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(0, 0, 0, 0); % "Ned"
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(0, 0, 1, 1); % "Op"
```

The probabilistic classifier is, opposed to the previously presented classifiers, 'soft' assignment, which means that each classification is given a probability value, e.g. "we are 92% certain this sample belongs to c_1."

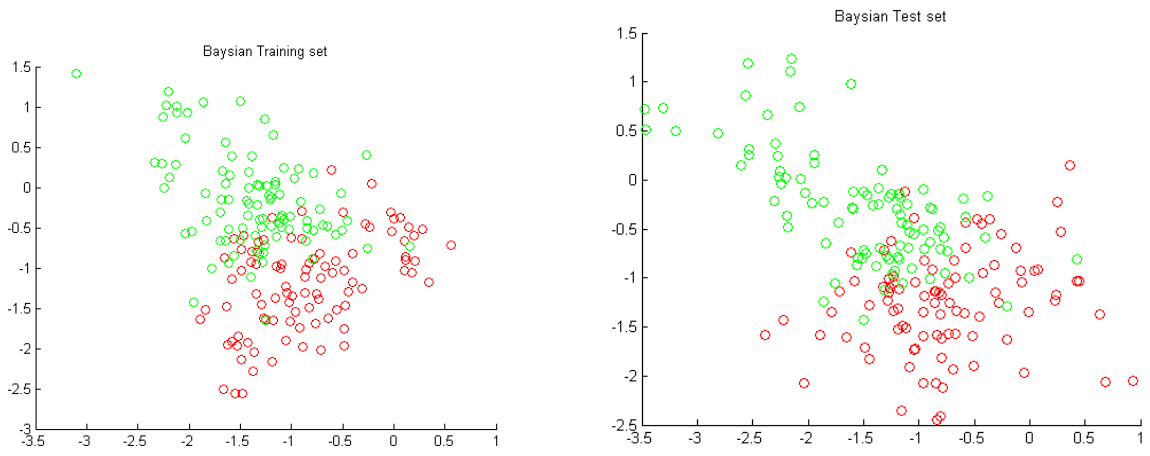
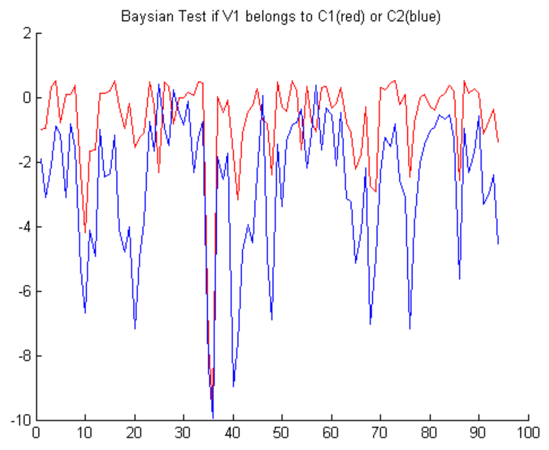


Figure 27 - Training and test set in 2D for "Op" recording



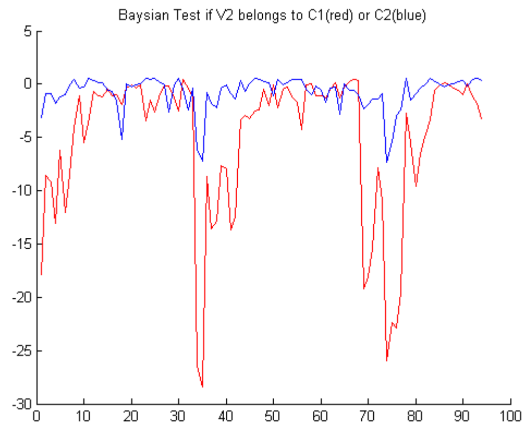
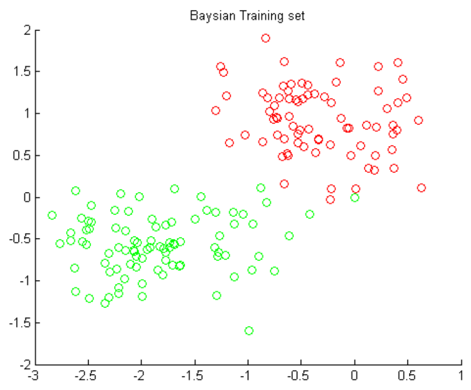


Figure 28 - Bayesian probability classification for voice 1 and voice 2 belonging to class 1 and 2 on test set

The Bayesian plots depict the logarithmic probability of V1 (Voice1) and V2 (Voice2) belong to one or the other class. We have attempted to classify into two classes, C1 containing V1 samples and C2 containing V2 samples. As we can see on the plots, the probability that a V1 sample belongs to C1 is generally higher than the probability it belongs to C2 and vice versa. This gives us the information necessary to make the classification.

$$Ctest = \begin{matrix} 89 & 19 \\ 5 & 75 \end{matrix} \qquad err_{test} = 0.13$$



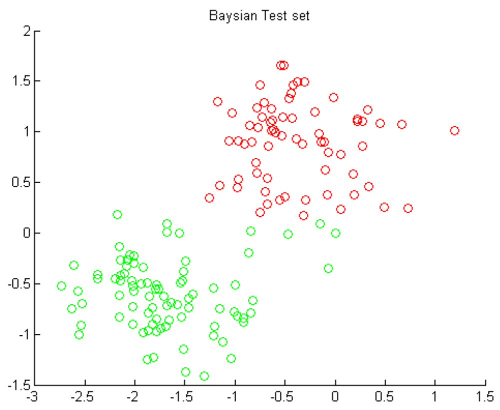


Figure 29 - Training and test set on MDA feature reduction in 2D for randomized "Ned" recording

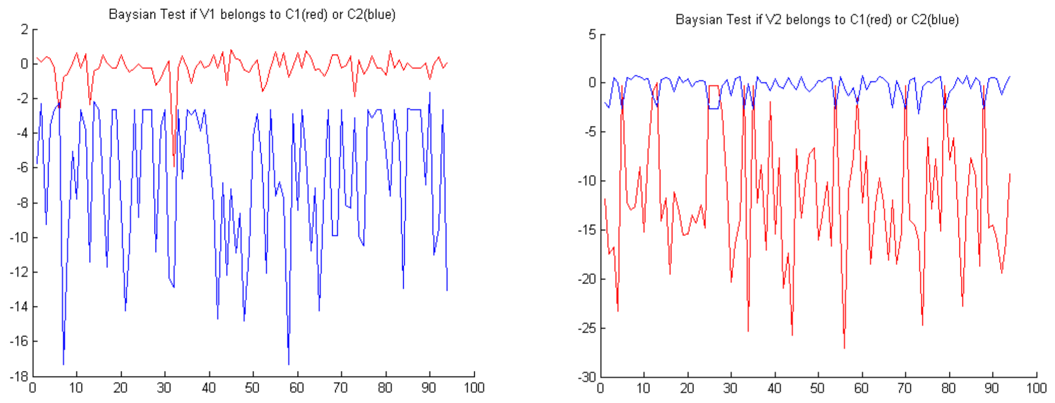


Figure 30 - Baysian probability classification for voice 1 and voice 2 belonging to class 1 and 2 for "Ned" recording

The result shows that we achieve a better result on the "Ned" recordings even though it is randomized, this is due to that the feature set of voice 1 and 2 are better separated.

$$Ctest = \begin{matrix} 93 & 13 \\ 1 & 81 \end{matrix} \qquad err_{test} = 0.07$$

Gaussian Mixture Model

```
UsePCA_MDAFeatureReduction = 2 % 0=none, 1=PCA, 2=MDA
UseClassificationMethodStart = 5
UseClassificationMethodEnd = 5
UseSizeTrainSet = 94
UseSizeTestSet = 94
UseRandomisation = 0
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(0, 0, 1, 1); % "Ned"

GMM2D.m - parameters

dimensions = 2;
ncentres = 3; % number of mixtures - try using e.g. 3, 5 and 7..
covartype = 'diag'; % covariance-matrix type.. 'spherical', 'diag' or 'full'
mix = gmm(dimensions, ncentres, covartype);

opts(3) = 0.001; % stop-criterion of EM-algorithm
opts(5) = 1; % do reset covariance matrix in case of small singular values.. (0=don't reset..)
opts(14) = 100; % max number of iterations
[mix, opts, errlog] = gmmem(mix, data, opts);
```

This chapter presents the results performing using Gaussian Mixture Models (GMM) for generative unsupervised classification. We have chosen to use MDA feature reduction in achieving a feature dimension of 2 using recordings from voice 1, 2 and silence as illustrated in Figure 32. The three feature set of these recordings are combined in one training set to evaluate unsupervised training in (GMM2D.m).

```
data = [Ynew(:,[1 2]); Wnew(:,[1 2]); Znew(:,[1 2])];
```

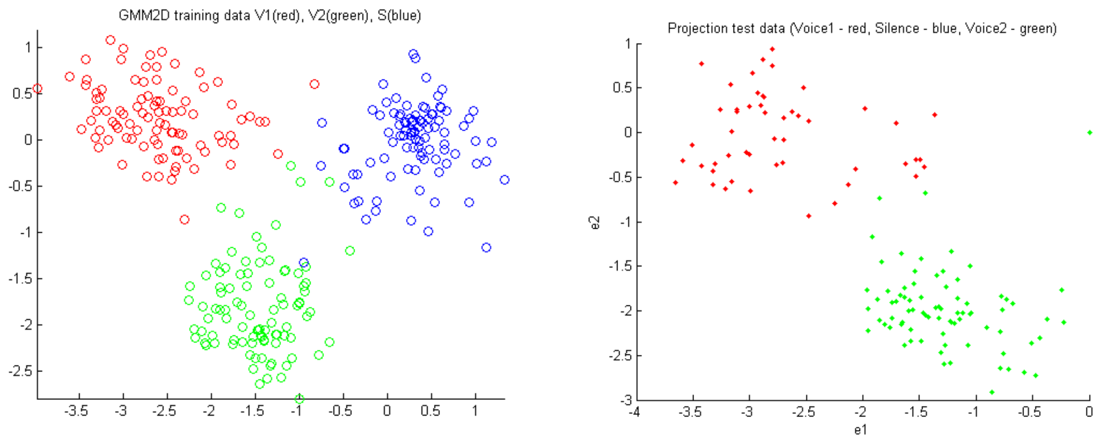


Figure 31 - Training and test set in 2D for "Ned" recording

Selecting a mixture of components where we have 3 Gaussian centers we are able to let the GMM algorithm automatically identify the 3 classes as illustrated in Figure 32. The covariance matrix is selected using the diagonal type as we can see in the resulting mixtures.

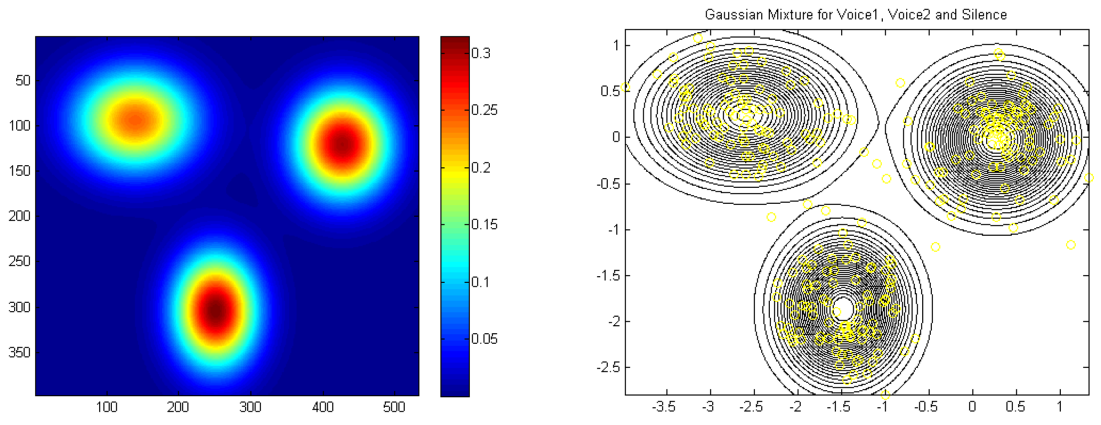


Figure 32 - GMM in 2D for unsupervised "Ned" recording

The resulting GMM mixture in MATLAB is listed below:

mix = type: 'gmm’, nin: 2, ncentres: 3, covar_type: 'diag', priors: [0.3246 0.3415 0.3339], centres: [3x2 double], covars: [3x2 double], nwts: 15

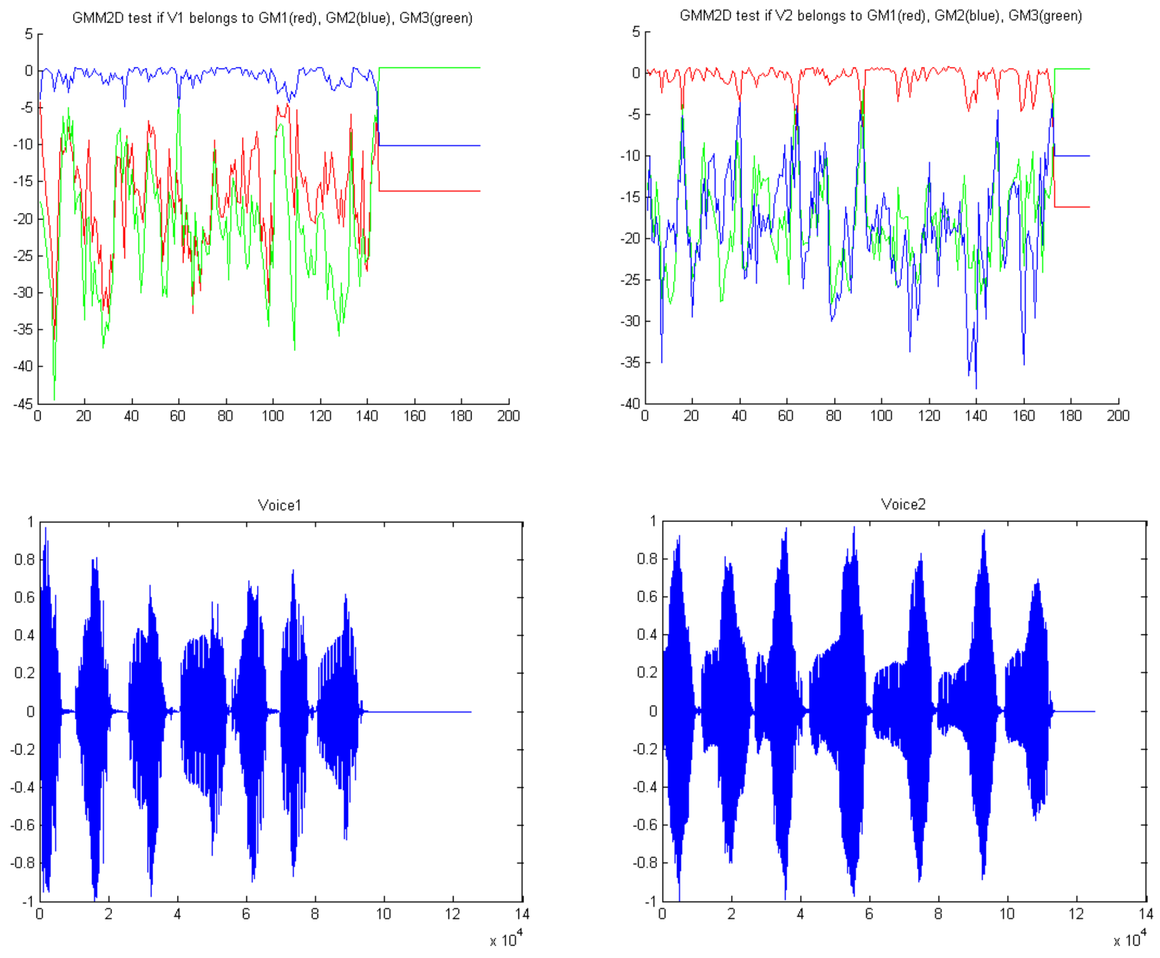


Figure 33 - Unsupervised GMM Classification of voice 1 and 2

The Bayesian plots depict the logarithmic probability of V1 (Voice1) and V2 (Voice2) belong to one or the other class. As this is unsupervised, we do not know which mixture corresponds to which class. However, the Bayesian plots clearly show that V1 samples correspond to the GM2 mixture, and V2 samples to the GM1 mixture. Based on the silence in the end of each recording, we can also see that GM3 probably represents the silence.

The confusion error we get on the test set is illustrated below.

$$C_{test} = \begin{matrix} 188 & 19 \\ 0 & 169 \end{matrix} \quad err_{test} = 0.05$$

As the Bayesian plots suggest, the classification error rate is very low. The columns of the matrix are V1,V2. The incorrectly classified V2 samples can be explained by the ~20 samples of silence in the end of the V2 recording. As we do not have a silence class, these are classified in either V1 or V2. The plots clearly show that silence samples are most likely to be V1 (blue line).

The classification method we have used is unsupervised. 5% error rate for an unsupervised learning is exceptional.

Trick question: Is our method truly unsupervised?

Answer: No. Before classifying, we reduced the feature space dimensionality using MDA. MDA is a supervised technique.

Going truly unsupervised by using PCA yielded error rates around 50%, which is the same as random guessing. Due to a high scattering of the voice samples and a high clustering of the silence samples, the mixtures were drawn towards the silence samples, not the voiced samples.

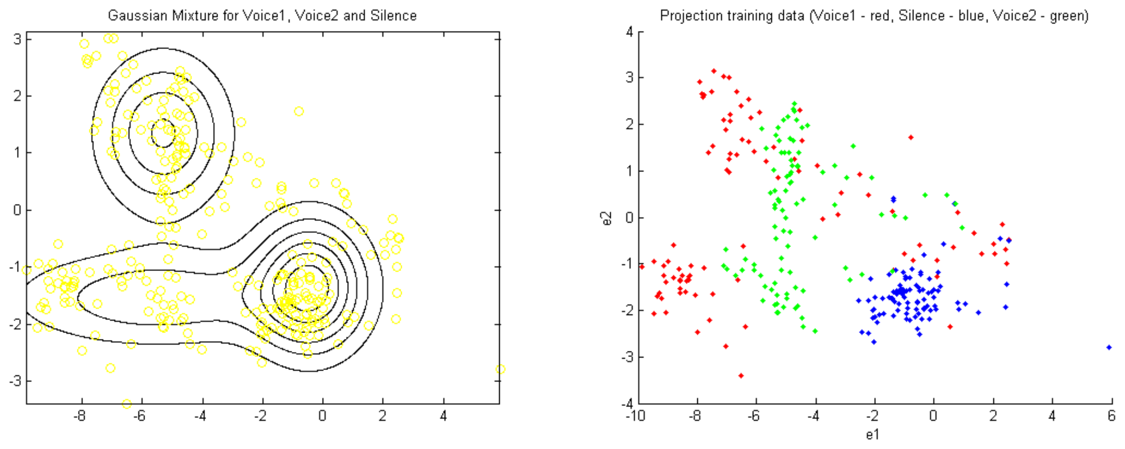


Figure 34 – Truly unsupervised PCA and GMM for "Ned" and silent samples

GMM and ANN on full set of recordings

The final experiment was to include all the recorded data of the two voices, including many different phonemes. Here we would like to conclude on comparing a generative and discriminative method used on all voice 1 and 2 recordings using MDA feature reduction.

```
UsePCA_MDAFeatureReduction = 2 % 0=none, 1=PCA, 2=MDA
UseClassificationMethodStart = 7
UseClassificationMethodEnd = 7
UseSizeTrainSet = 1700
UseSizeTestSet = 100
UseRandomisation = 1
[mfcc_voice1 mfcc_voice2 mfcc_silence] = CreateMFCCSamples(0, 0, 0, 5); % "All recordings"

………

[Ctrain, Ctest] = GMM2DComponents(V1new, V1tnew, V2new, V2tnew, 5); % ncentres = 5

GMM2DComponents.m - parameters

dimensions = 2;
covartype = 'diag'; % covariance-matrix type.. 'spherical', 'diag' or 'full'
mix = gmm(dimensions, ncentres, covartype);

opts(3) = 0.0001; % stop-criterion of EM-algorithm
opts(5) = 1; % do not reset covariance matrix in case of small singular values.. (1=do reset..)
opts(14) = 100; % max number of iterations
```

First we will try to use the GMM in combination with supervised learning. We have extending our training to cover all recordings including in total 1700 samples as plotted in Figure 35. A Gaussian mixture model with 4 components is selected. Training is performed for each known class that includes the voice 1 and voice 2 of the sample feature set. A test set of 150 samples are used for generative classification using the 4 Gaussian components from each class. The Bayesian formula is used in finding the mixture of 2x4 that has the highest likelihood that the test sample belongs to.

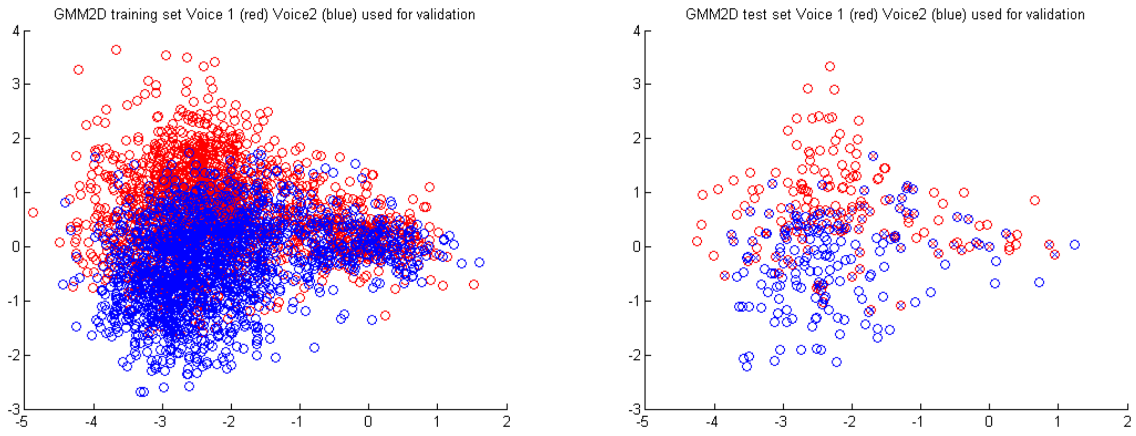


Figure 35 - Test and training data randomized from all recordings

In Figure 35 we see the training data set and the result of classification on the test set. Samples with incorrect classification are marked with an X.

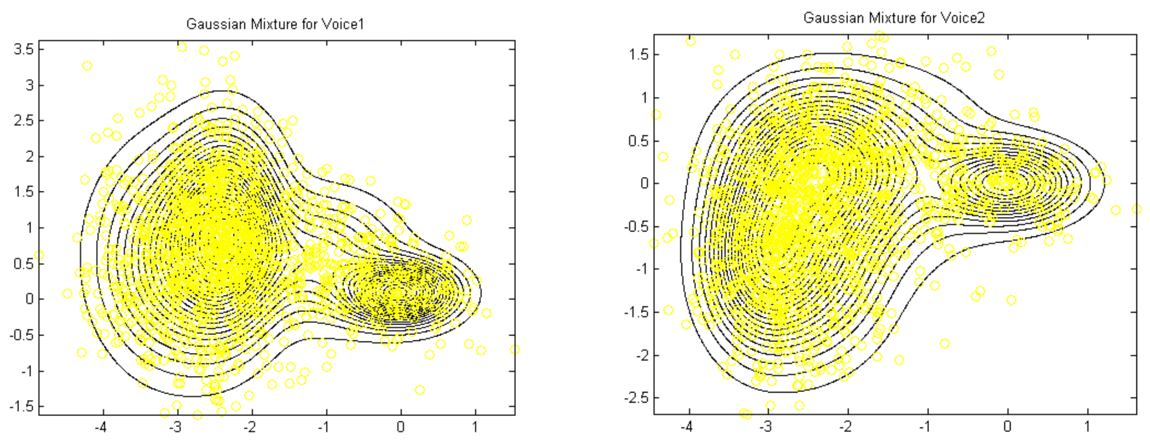


Figure 36 - Supervised GMM 2D Classification of voice 1 and 2

By this method we get a classification error of 0.28.

$$C_{test} = \begin{matrix} 99 & 33 \\ 51 & 117 \end{matrix} \quad err_{test} = 0.28$$

Finally we are validating the trained GMM 2D classification method on speaker V1 and V2 reading a specific sentence. We can see from the plots that the likelihood changes as function of sample number. Each sample number on the X-axis represents a step of 15 ms of audio. For the 377 samples we get a recording of total 5.6 sec.

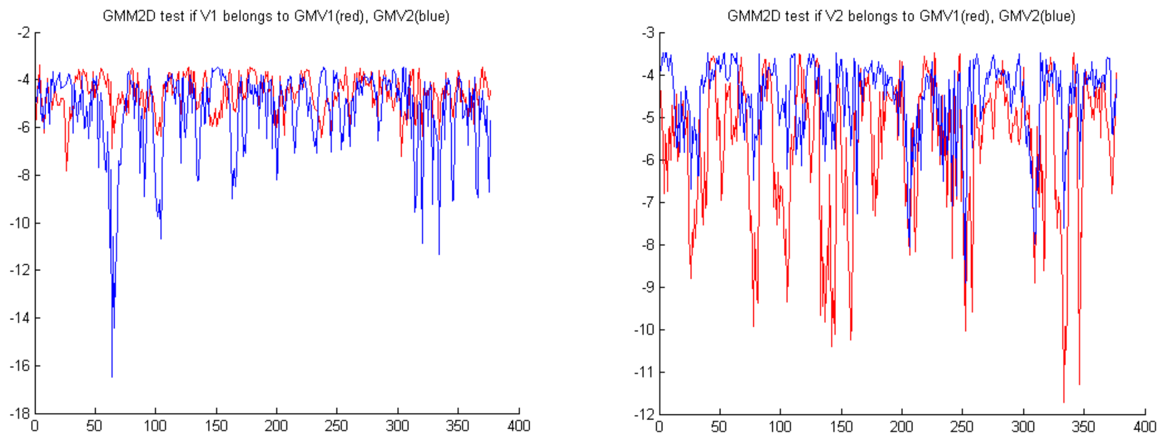


Figure 37 - Classification for voice V1 and voice V2 reading "Sentence 1"belonging to the Gaussian Mixture (GMV1 and GMV2)

We see that it is should be possible to give an estimate of who is speaking and thereby recognition of the speaker, by averaging the red and blue curves over a time window.

The error we get reading "Sentence 1" is show below nearly the same as for randomized data.

$$C_{test} = \begin{matrix} 247 & 94 \\ 130 & 283 \end{matrix} \quad err_{test} = 0.29$$

If we use the same setting as described above for the ANN 2D method we get nearly the same error.

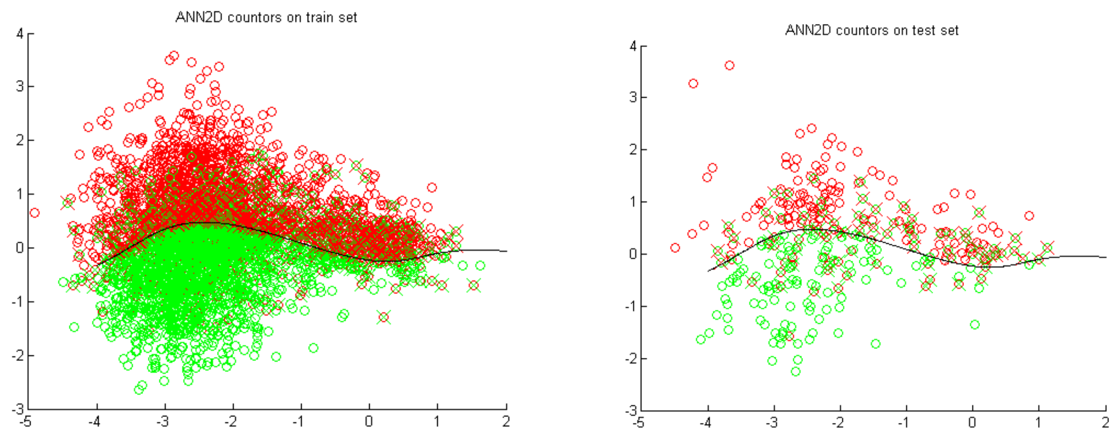


Figure 38 - Supervised ANN 2D Classification of voice 1 and 2

$$C_{test} = \begin{matrix} 113 & 37 \\ 48 & 102 \end{matrix} \quad err_{test} = 0.28$$

Discussion of Results

In this chapter we will reflect over and discuss the results produced and the lessons learned

Summary of our results combining different techniques on non randomized samples

Method	*Reduction*	*Train "Ned"*	*Test "Ned"*	*Train "Op"*	*Test "Op"*
0.Linear 2D	PCA	0.26	0.48	0.45	0.39
0.Linear 2D	MDA	0.01	0.23	0.14	**0.16**
1.Linear 3D	PCA	0.44	0.52	0.42	0.52
2.ANN 2D	PCA	0.14	0.20	0.14	0.21
2.ANN 2D	MDA	0.00	0.25	0.13	**0.16**
3.ANN 6D	PCA	0.004	0.27	0.04	**0.11**
4.Probalistic 2D	PCA	-	0.20	-	0.27
4.Probalistic 2D	MDA	-	**0.07**	-	**0.13**
5.GMM 2D	PCA	-	0.51	-	0.61
5.GMM 2D	MDA	-	**0.05**	-	0.46
6.GMM 3D	PCA	-	0.37	-	0.57
7.GMM 2D Comp	PCA	-	0.34	-	**0.18**
7.GMM 2D Comp	MDA	-	**0.09**	-	**0.16**

Additional to this, we performed a MDA to reduce a large data set of same voices pronouncing a full sentence. We then used ANN 2D and GMM 2D to classify the data and both methods resulted in ~28% error rate.

Supervised or unsupervised?

Supervised training methods are techniques that require a prior knowledge of the data to be analyzed, for example to determine a decision boundary that can be applied when new samples are recorded. Unsupervised training methods are techniques that can be performed on a data set without prior knowledge of the data.

The supervised techniques we have tried in this project were Multiple Discriminant Analysis for feature reduction, Artificial Neural Networks, Linear Regression Classifier and Bayesian Probabilistic Classifier for the classification. The unsupervised techniques we tried were Principle Component Analysis for feature reduction, and Gaussian Mixture Model for classification.

Comparing feature reduction techniques, MDA and PCA, MDA has an extra constraint in that it can only project to *c-1* directions for a *c*-class problem. We had 3 classes, so we could not project to more than a 2D feature space. PCA is not limited in feature space dimensionality. To investigate the effect of a higher feature dimensionality, we can compare the ANN 2D And ANN 6D results; the training classification for 6D becomes very good, probably due to the extra degrees of freedom. However, the effect on the test set is questionable.

When comparing the plots and classification results for the feature reduction techniques, we can see that MDA is generally superior for classifying data and should be the preferred procedure for dimensionality reduction whenever possible when it comes to supervised classificatiion.

When comparing the classification methods for MDA, we note a fairly good performance of the PBC and ANN classifiers, both supervised techniques. But we also see surprisingly good performance for the unsupervised GMM classifier. However, GMM requires a good clustering of the samples to perform. When we attempted GMM on a more scattered set (Figure 34) the mixtures were gathered in the most clustered, but unfortunately least interesting part of the plot: The silence area. This led to very poor performance of the classifier for voice identity. Since PCA doesn't perform as well for clustering data, a truly unsupervised training yielded error rates no less than 37% even when only a few phonemes were used. This is too high for reliable speaker identification.

Data Set

When choosing the data sets for the project we had one concern: We use MFCC to extract features. Voices have different characteristics in the MFCC, but so do the phonemes. When trying to separate samples based on the most significant MFCC features, can we avoid classifying based on the text content of the recording?

In our experience, PCA was not suited for MFCC feature extraction. The problem of using PCA for speaker identification was, most of the variance produced in a data set does not stem from different speaker identities, but from the content of what is being said and so, we end up clustering the data by phoneme rather than by identity. When using PCA for sets with more than one phoneme it becomes hard to reliably separate the speaker identities.

For MDA we were able to separate the samples by speaker identity even with a high variance in phonemes. However, classification error rate increased with increased variance, due to a larger overlap between the two clusters voice clusters caused by phonetic variance.

MFCC feature extraction is usually utilized when performing speech recognition, the process of translating voice recording into text. When we decided to use it for speaker recognition in the first place, it was mostly due to the fact that it was a proven method for generating features in a speech-related scenario. Other speech-related feature extraction techniques include Linear Predictive Coding (LPC), but this was not explored in this project.

Future Work

Our validation set of recordings is very narrow – only two voices, both from adult males. The set should be extended to cover voices from different female and children in ensuring a better validation of our classification methods. We would be able to increase the number of dimensions for MDA feature reduction. Cross-validation could also be applied to improve the confidence level of our presented results.

The variability from different speakers is related to different vocal cords and the vocal tract. Different speakers are not producing the same acoustic signal. Typically, females sound is different from males. So do children from adults. The challenge is to find features that characterize the personality of the acoustic signal. In our work we have focused on the Mel-cepstrum (MFCC). The alternative LPC (Linear Predictive Coding) is also widely used in speech recognition. The main idea behind linear prediction is to separate the excitation spectrum from the vocal system spectrum. The excitation spectrum is responsible for the “fast” spectral variations and the vocal system spectrum is responsible for the “slow” spectral variation. The main idea behind LPC is to extract the vocal tract parameters by modeling the vocal tract filter in where we find the LPC coefficients used as features for classification.

The prospect of using ANN and GMM models for speaker classification is promising. It would have been interesting to extend the GMM method with 4 components for each voice with a higher dimension than two dimensions. Would it be possible to achieve a better accuracy with the PCA feature reduction and 6 dimensions?

In our current model we are not taking into account that the samples are dependent on each other as function of time (**Figure 37**). As a consequence of this, we expect the system to identify a speaker based on 30ms of recording. Not even humans are capable of this feat! If a sample would be classified as belonging to one voice, each neighbouring sample would have a high probability of belonging to the same voice, due to the fact that humans rarely talk for only 30ms at a time. Since the human brain is very good at voice classification, but unable to tell for sure who is speaking based on 30ms of voice, the brain clearly uses a larger time frame to identify who the speaker might be. We could improve the classification hit rate by introducing a model that incorporates a greater dependency between samples, such as the Markov models or a low pass filter.

To raise the hit rate even more, a combination of multiple independent systems could be used. The systems would be trained independently and vote for the result, lowering the probability of error.

The use of Hidden Markov Models (HMM) in speaker classification would also enable us to create a text-dependent speaker recognition system. HMM is good for classification a sequence of patterns. Would we be able to distinguish between speakers reading the same sentence if we trained a HMM for each speaker?

The paper [4] presents a method for speech recognition where they have investigated the combined used of MFCC and LPC features. This approach seems to approve the reliability of the speech recognition system. They use training and recognition realized by artificial neural networks (ANN). The MFCC and LPC-based recognition are combined as two independent recognition processes (ANN) realized in parallel. The recognition results of MFCC and LPC-based are compared and the speaker is confirmed to be identified if confirmed by both subsystems. A similar method could be investigated using a generative probabilistic model as described in this report or extended with more subsystems in parallel using a combination of generative and discriminative classification methods.

Conclusion

The goal of this project was to apply techniques of dimensionality reduction and machine learning taught in the course ‘Non-Linear Signal Processing and Pattern Recognition’, TINONS, in order to investigate if we were able to classify different speaker identities.

The methods for machine learning included Mel Frequency Cepstral Coefficients for feature extraction, Multiple Discriminant Analysis and Principal Component Analysis for feature dimensionality reduction, and Linear Classification, Probabilistic Bayesian Classifier, Artificial Neural Networks and Gaussian Mixture Model for sample classification. Based on recordings of a single word with only a few phonemes pronounced by two grown male voices, we were able to classify the speaker identity with an error rate of 5-15% in cases where we had prior knowledge of the data. In cases where we had no prior knowledge of the data we were unable to distinguish speaker identity. Based on recordings of whole sentences – many phonemes, pronounced by the same voices, we were able to identify the speaker with an error rate of 28%, given we had prior knowledge of the data set. We expect the inclusion of time dependency, by means of filtering or Markov models, to decrease the classification error rate.

References

[1] Davis and Paul Mermelstein. Steven B. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(4):357–366, 1980.

[2] M. H. Hayes. *Statistical Digital Signal Processing and Modeling*. J. Wiley & Sons, Inc., New York, 1996.

[3] Richard O. Duda, Peter E. Hart, David G. Stork, *Pattern Classification*, Wiley, 2001, ISBN 978-0-471-05669-0

[4] K.R. Aida-Zade, C. Ardil and S.S. Rustamov, *Investigation of Combined use of MFCC and LPC Features in Speech Recognition Systems*, World Academy of Science, Engineering and Technology, 2006