TINONS Miniproject

Speaker Recognition

**<date>**

Kim Bjerge (KBE / 20097553)

Bjarke Møholt (20041658)

Contents

[Introduction (Kim) 3](#_Toc325471324)

[Theory (Kim) 4](#_Toc325471325)

[Mel-cepstrum (Kim) 5](#_Toc325471326)

[Principal Component Analysis (Bjarke) 7](#_Toc325471327)

[Multi-Discriminant Analysis (Bjarke) 8](#_Toc325471328)

[Linear Regression Classifier (Bjake) 9](#_Toc325471329)

[Bayesian Classifier / probabilistic classifier (Bjarke) 10](#_Toc325471330)

[Artificial Neural Networks (Kim) 11](#_Toc325471331)

[Gaussian Mixture Model (Kim) 15](#_Toc325471332)

[Conduct of Experiments 17](#_Toc325471333)

[Results 21](#_Toc325471334)

[Principal Component Analysis 21](#_Toc325471335)

[Multi-Discriminant Analysis 24](#_Toc325471336)

[Linear Regression Classifier 24](#_Toc325471337)

[Bayesian Classifier/ probabilistic classifier 26](#_Toc325471338)

[Artificial Neural Networks 29](#_Toc325471339)

[Gaussian Mixture Model 30](#_Toc325471340)

[Discussion of Results 31](#_Toc325471341)

[Conclusion 32](#_Toc325471342)

# Introduction (Kim)

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 have developed are text-independent, meaning the system can identify the speaker regardless of what is being said. Pattern classification methods typical contain two functionalities: A training mode and a recognition mode. The training mode will allow the user to record voice and performs feature extraction used for pattern classification. The recognition mode will use the information that the user has provided in the training mode and attempt to isolate and identify the speaker based on distinguishing features. 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 algorithm based on supervised and unsupervised learning presented in the course TINONS. The algorithms presented in this report cover both discriminative and generative models like Artificial Neural Networks and Gaussian Mixture Models. 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.

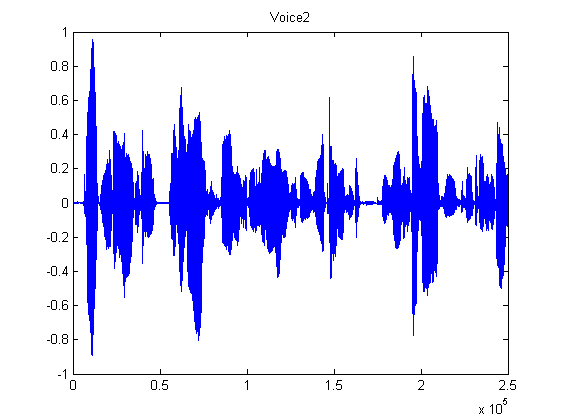


Figure 1 Recording from speaker reading a sentence

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.

# Theory (Kim)

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]. The theory 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 either the Principal Component Analysis (PCA) or 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 and optimal 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 Baysian 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 Models (GMM). GMM are 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. Would it be possible to determine who is speaking without supervised training? Hidden Markov Models are not investigated in this project and therefore the theory is not described. (TBD or only briefly described)

## Mel-cepstrum (Kim)

A range of possibilities exist for parametrically representing the features of the speech signal for the speaker recognition, such as Linear Prediction 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. 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. A 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 Hanning 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.

Mel (f) = 2595\*log10 (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

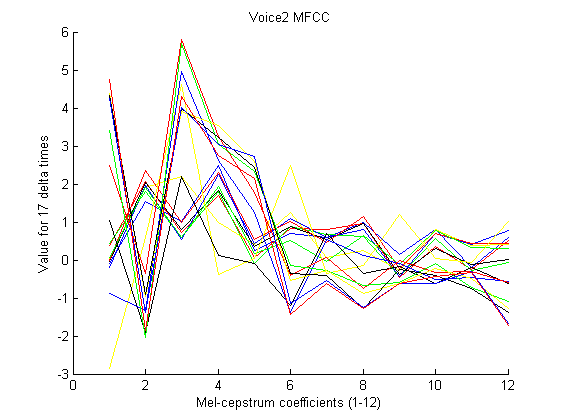


Figure 3 Mel-Frequency Cepstrum Coefficients for 17 frame blocks

## Principal Component Analysis (Bjarke)

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 x1…xn by a single vector x0, so that the squared distances between xk and x0 are as small as possible. The squared error function is then

We want to minimize this. This is a trivial 0-dimensional representation of the data set, and it can be shown that *x0* = **m** is the minimizer, where m is the sample mean. 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

where **e** is a unit vector in the direction of the line. If we then represent *xk* by ,we can find an optimal set of coefficients ak by minimizing the squared error, also known as the cost function:

Remembering |**e**|=1, partially differentiating with respect to *ak* and setting the derivative to zero, we obtain

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

It appears obvious that the **e** that minimizes *J1* also maximizes **e**t**Se**, under the constraint that |**e**|=1. To maximize **e**t**Se**, we use the method of Lagrangian multipliers and get:

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.

## Multi-Discriminant Analysis (Bjarke)

PCA is useful for finding greatest variance and thus, representing data. However, finding the best direction for *representing* data is not necessarily the same as finding the best direction for *discriminating* data. The Fisher Linear Discriminant (FLD) method has been proposed for this purpose, and Multi-Discriminant Analysis (MDA) is a generalization of the FLD for more than 2 classes.

FLD begins by supposing we have a set of *n* samples **x***1*, …, **x***n*. These samples are divided into the subsets *D1* and *D2*. The subsets *Y1* and *Y2* are obtained from **x** by a linear projection *y =* **w**t**x**, where **x** ϵ *D1* and *D2* 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 **m**i be the sample mean given by

then the sample mean for the projected points is given by

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

## Linear Regression Classifier (Bjake)

## Bayesian Classifier / probabilistic classifier (Bjarke)

## Artificial Neural Networks (Kim)

Multilayer Perception (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 applies 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 and output layers 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.

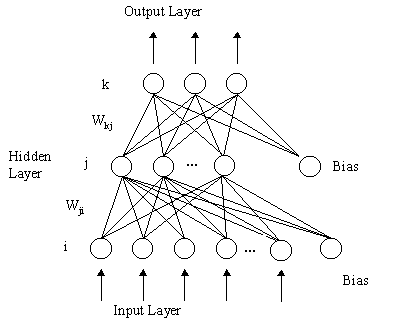


Figure 4 Multilayer Neural Network with three-layers

The output discriminant functions can be expressed as

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 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 activation function ensures values between 0-1 and good for 2 classes

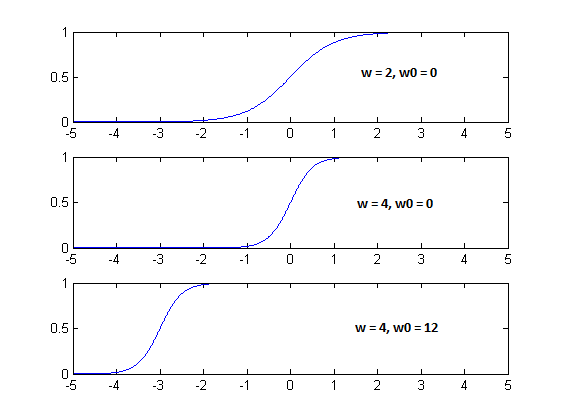


Figure 5 Logistic activation function with variation of w (slope) and w0 (offset)

The Sigmoid activation function

The generalized softmax activation function for output where and , and good for more than two classes

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 can define a criterion function that needs to be minimized in finding the best MLP network.

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 () is adjusted to impose the regularization more or less strongly. The solution for 2 classes we get the *cross-entropy* cost function.

Training is very time consuming and it is difficult to automate since the training for an optimal set of variables are dependent on each other. 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 6 where we have two classes C1 and C2 with a Gaussian distribution and a region of overlap that defines the minimal theoretical error.

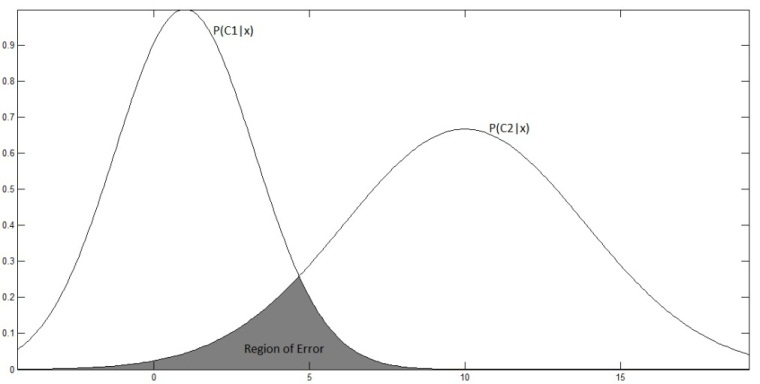


Figure 6 Bayes error for overlapping densities

The error is typical high before the training has begun. Through learning the error becomes lower, as shown in the learning curve (Figure 7). 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.

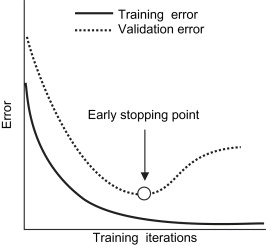
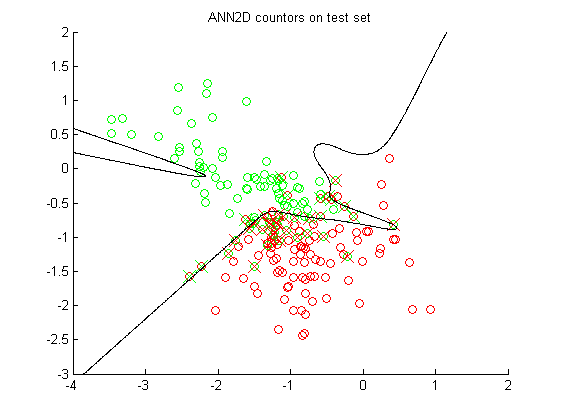
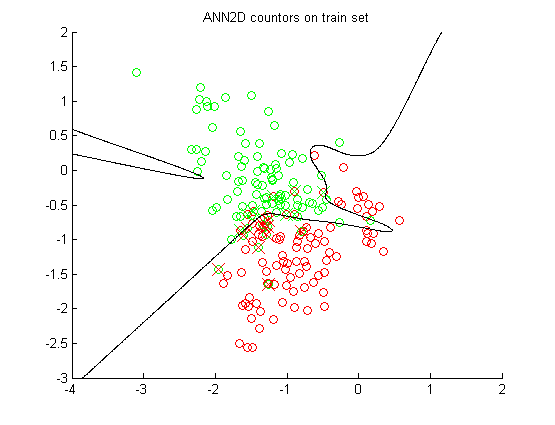
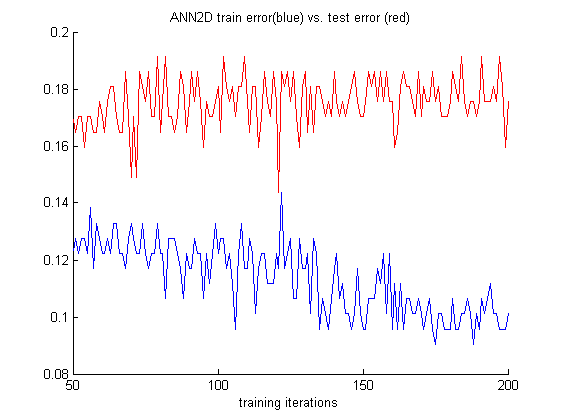


Figure 7 Learning and validation curves

The problem we get is over fitting 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.





## Gaussian Mixture Model (Kim)

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, to compute and accelerate the convergence of finding clustering patterns in the sample data, is 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 for each cluster
3. Assign sample points to each mean cluster
4. Calculate new mean values: , where is the responsibility whether sample **n** belongs to the given cluster *i*.
5. Iterate point 3+4 until no change in the means or that a defined cost function based on the squared Mahalanobis distance is small. The cost function is

If we assume that we know the complete probability structure for the problem. We have to find the unknown parameters of the probability distribution and the number of patterns (*k*). Assuming that the distribution for each pattern is Gaussian we need to find the unknown variance and mean parameters:

The probability density function for the samples is then given by

where and *k* are the unknowns. The conditional densities are called *component densities*, and the prior probabilities are called the *mixing parameters*. For the Gaussian Mixture Model (GMM) the conditional densities are the multivariate normal distribution.

where ∑ 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 , diagonal where and full where ∑ could be anything. To find the optimal solution to GMM we will use the maximum-likelihood to estimate the unknown parameters similar to find the maximum of the log-likelihood given by

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

here the mixing coefficient are called , we have

where is a normal distribution. We can write our solution formally as the argument that maximizes the log-likelihood

In finding the maximum we need to differentiate the log-likelihood according to the parameters: to find:

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 of the parameters . We used then the Bayes formula to compute the probability for the samples belonging to class in the E-step

where is the normal distribution. In the M-step we compute new estimates for the means , covariance matrix’s and mixing coefficients .

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 a maximum number of iterations are done. The algorithm is sensitive to estimates of the covariance matrix. When the eigenvalues becomes very small the value of goes to infinite, therefore the eigenvalues () 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 have developed 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. The recordings are recording with the audio/MIDI-sequencer Cubase from Steinberg. All recordings are normalized and stored in wave files with a sampling rate of 44.1 kHz. The recordings contain speech from a male voice 1 and 2. The recordings contain different scenarios. A single word repeated a number of times, where the silence part is removed. Different sentence recorded one or more times. The length of each recording is between 3 – 6 seconds.

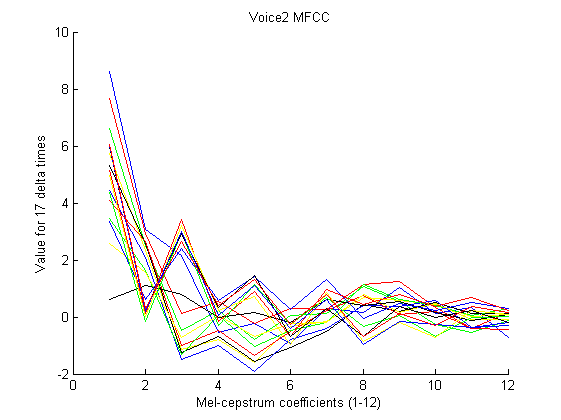
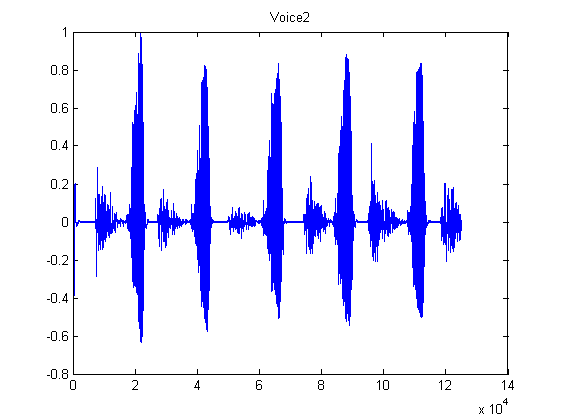
## Audio recordings

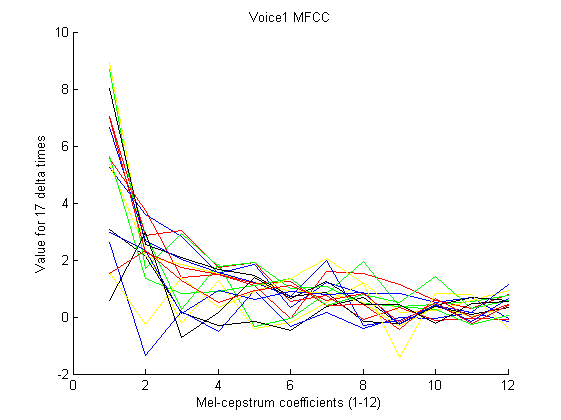
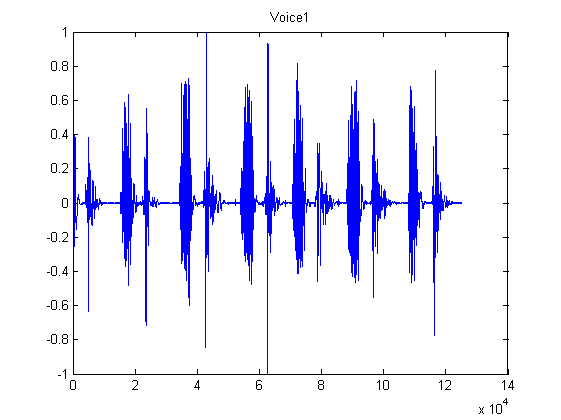
A MATLAB function is made that creates the sample features set for the recordings. With the CreateMFCCSamples function it is possible to specify the type of recordings from where the sample feature set should be generated. The result is a sample set for voice 1, 2 and silence, containing a feature set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

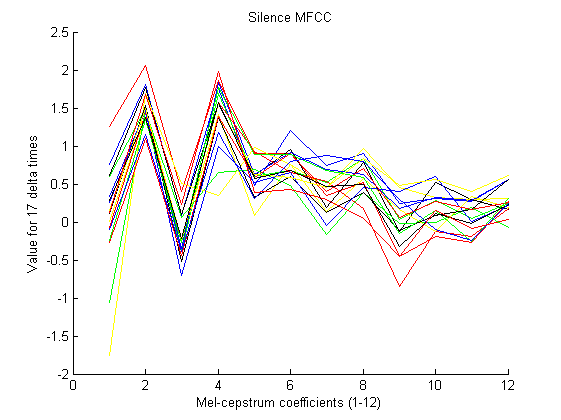
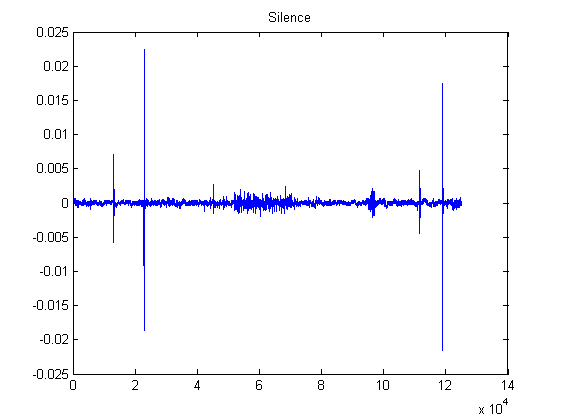
function [mfcc\_voice1, mfcc\_voice2, mfcc\_silence] = CreateMFCCSamples(PlotMFCC, Pause, Start, End)

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. 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”. | 188 | 0 |
| NedBjarkeC.wav, NedKimC.wav, Silence.wav | Recordings of speaker voice pronouncing “Ned”. | 188 | 1 |
| Speech1\_1.wav, Speech2\_1wav, Silence2.wav | Reading sentence 1 | 377 | 2 |
| Speech1\_2.wav, Speech2\_2wav, Silence2.wav | Reading sentence 1 (Same as above) | 377 | 3 |
| 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” 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. 250 ms. The sample rate is 44.1 kHz.

## MATLAB program

All experiments that explore the different algorithms described in this report are combined in one MATLAB program. The program calls the functions that implement the different algorithms to generate training and test data sets, performing feature reduction, plotting the sample feature space and finally using the different classification methods and algorithms. A number of parameters can be set to specify the execution of the program (VoiceClassificationAllRand.m) as listed 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

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) methods. The classification methods can be specified 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 |
| 6 | Performs unsupervised classification using the Gaussian Mixture Model on the training set of 3 dimensions or higher |

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

linear2D\_train = 1-sum(diag(Ctrain))/sum(Ctrain(:)); % correct classification percentage

linear2D\_test= 1-sum(diag(Ctest))/sum(Ctest(:)); % correct classification percentage

case 1

% 3D classification training set with 2 classes and 3 or more features

[Ctrain, Ctest, W] = linear3D(V1new, V1tnew, V2new, V2tnew); % training

linear3D\_train = 1-sum(diag(Ctrain))/sum(Ctrain(:)); % correct classification percentage

linear3D\_test= 1-sum(diag(Ctest))/sum(Ctest(:)); % correct classification percentage

case 2

% 2D classification using Artificial Neural Networks

[Ctrain, Ctest] = ANN2D(V1new, V1tnew, V2new, V2tnew, Snew, Stnew, 2); % 2 or 3 features

ANN2D\_train = 1-sum(diag(Ctrain))/sum(Ctrain(:)); % correct classification percentage

ANN2D\_test= 1-sum(diag(Ctest))/sum(Ctest(:)); % correct classification percentage

case 3

% 3D classification using Artificial Neural Networks

[Ctrain, Ctest] = ANN3D(V1new, V1tnew, V2new, V2tnew, Snew, Stnew, size(subSet,2));

ANN3D\_train = 1-sum(diag(Ctrain))/sum(Ctrain(:)); % correct classification percentage

ANN3D\_test= 1-sum(diag(Ctest))/sum(Ctest(:)); % correct classification percentage

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

gausianDiscriminant\_test= 1-sum(diag(Ctest))/sum(Ctest(:)); % correct classification

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);

GMM2D\_test= 1-sum(diag(Ctest))/sum(Ctest(:)); % correct classification percentage

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));

GMM3D\_test= 1-sum(diag(Ctest))/sum(Ctest(:)); % correct classification percentage

otherwise

% Invalid classification parameter specifier

end

# Results

* Plotting in 2D for premilary analysis
* Confusion matrix
* Reading guide

## 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”



Figure 8 PCA feature reduction in 2D for "Ned" recording

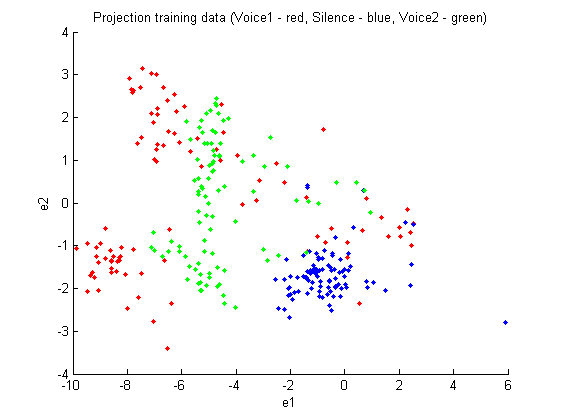


Figure 9 PCA feature reduction in 2D for "Op" recording



Figure 10 PCA feature reduction of randomized "Op" training data

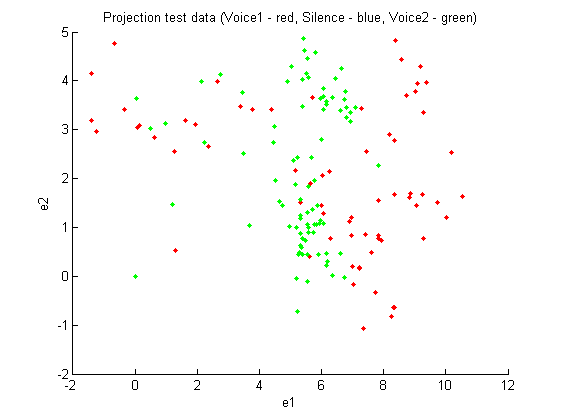


Figure 11 PCA feature reduction of randomized ”Op” test data

## Multi-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”

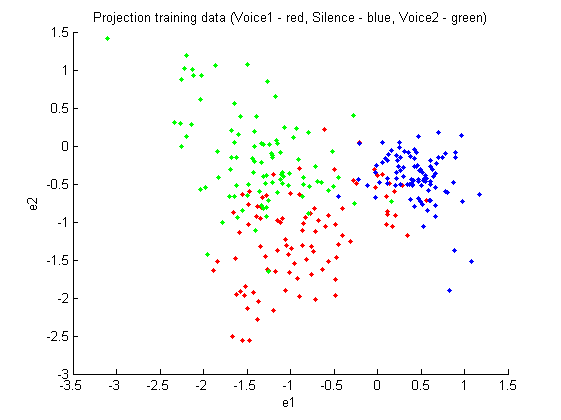


Figure 12 MDA feature reduction in 2D for "Op" recording



Figure 13 MDA feature reduction in 2D for "Ned" recording

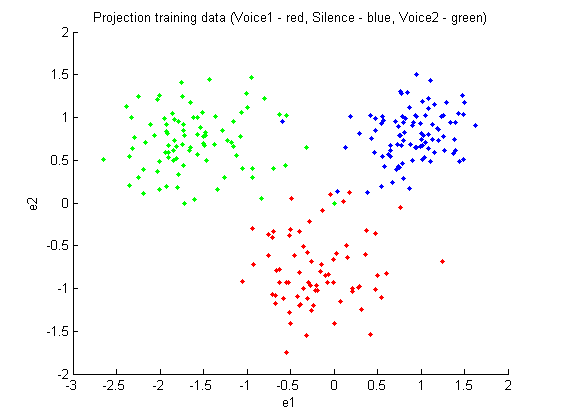


Figure 14 MDA feature reduction in 2D for randomized "Ned" recording

## Linear Regression 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”

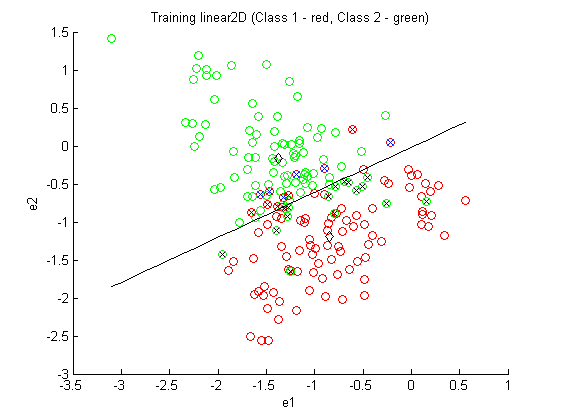


Figure 15 Training for linear classification on MDA feature reduction in 2D for "Op" recording



Figure 16 Test for linear classification on MDA feature reduction in 2D for "Op" recording

Ctrain =

83 16

11 78

err\_train = 0.1436

Ctest =

89 25

5 69

err\_test = 0.1596

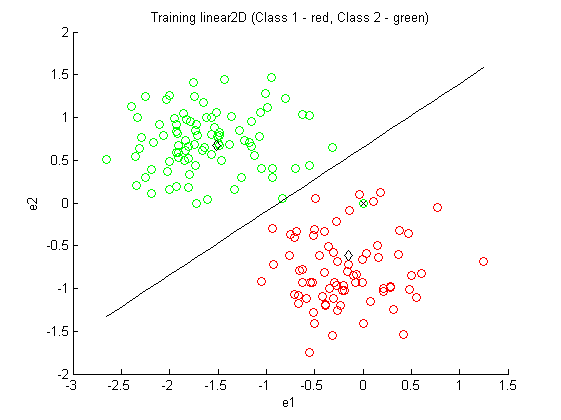
****

Figure 17 Training for linear classification on MDA feature reduction in 2D for randomized "Ned" recording

****

Figure 18 Test for linear classification on MDA feature reduction in 2D for randomized "Ned" recording

Ctrain =

94 6

0 88

err\_train = 0.0319

Ctest =

94 13

0 81

err\_test = 0.0691

## 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”

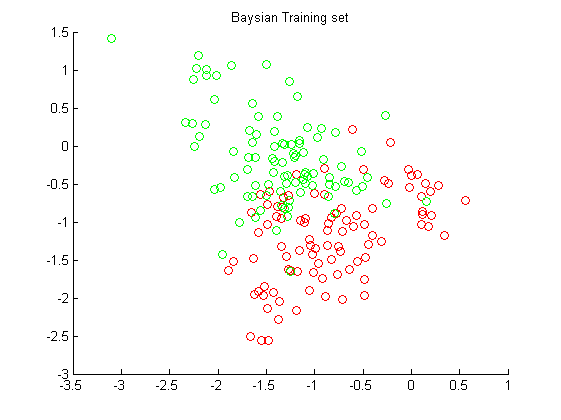
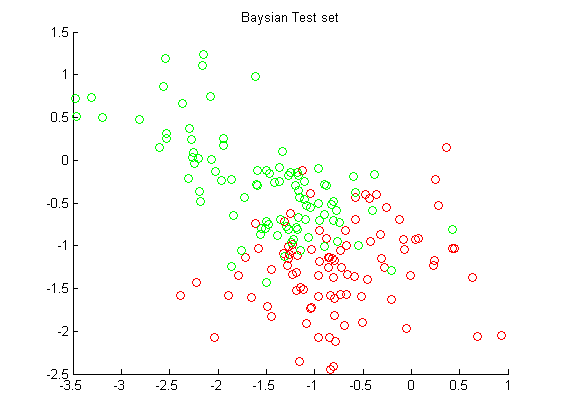


Figure 19 Training and test set in 2D for "Op" recording

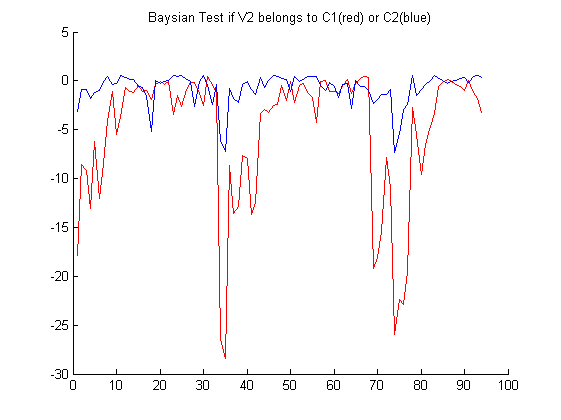
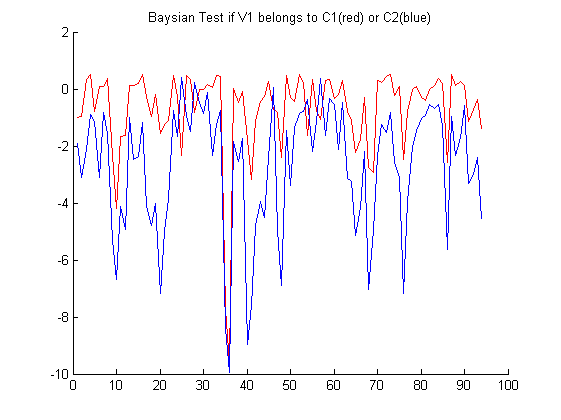


Figure 20 Baysian probability classification for voice 1 and voice 2 belonging to class 1 and 2 on test set

Ctest =

89 19

5 75

err\_test = 0.1277

## Artificial Neural Networks

## 

## Gaussian Mixture Model

# Discussion of Results

# Conclusion

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