# Speaker Recognition in non-linear signal processing and pattern recognition.

Rune A. Heick, Rasmus S. Reimer & Nicolai Glud

Aarhus University, Department of Engineering

**Abstract**

*The Content of this paper seeks to present the knowledge gained throughout the non-linear signal processing and pattern recognition course from Aarhus University, department of engineering. The paper is split into multiple sections explaining the data used in the paper, the methods used to treat the data and the methods used for categorising the data.*

## I. Introduction

The idea behind the project is to recognise the speaker using the methods and categorisers learned in the course pattern recognition and machine learning (TINONS). The voices of all authors was recorded and imported to matlab. The features from the data was extracted in matlab using the Mel-frequency cepstral coefficient(Hereafter MFCC) method from the voicebox toolbox. The MFCC's are used as features for the classifiers that are tested in this paper.

## II. Data Gathering

The data used in this project was gathered by recording three different persons reading the same article from the website "www.tv2.dk". The voices was recorded using the software Audacity[1] and the Lame mp3 codex[2]. The data is then imported into matlab using the function `[data, Fs] = audioread(pathToFile)`. The data is then normalised by removing the mean of the data, and whitening the data. The files are in stereo and both channels are used by appending one channel to the other so to have one long array of data.

## III. Feature Extraction

The Mel Frequency Cepstral Coefficient method is commonly used to extract features in speech and speaker recognition. The methods provides features that are useful for classifying linguistic content. The basis is that the speech from humans is uniquely filtered by the shape of the vocal tract, tongue, teeth etc[3].
MFCCs are found using a series of steps as can be seen in figure 1.

[1]http://sourceforge.net/projects/audacity/
[2]http://lame.sourceforge.net/
[3]http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/ - retrieved 4 june 2015

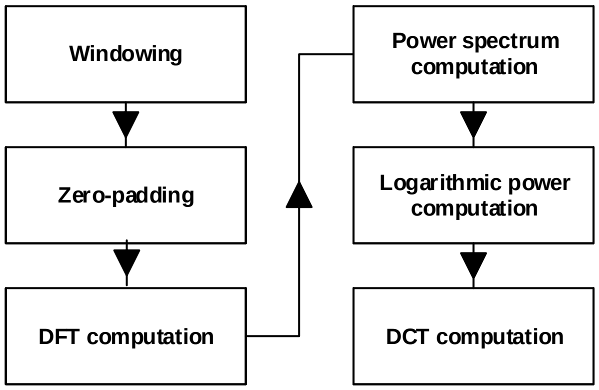

**Figure 1:** *MFCC steps*

The figure has been derived from the article [**?**]. The steps can be described as follows:

1. Windowing: The speech signal is windows with either a Hamming or Hanning window.
2. Zero-padding: A number of zeros are padded to he windowed speech signal in order to enable fast Fourier transform (FFT).
3. DFT: The windowed speech signal is discrete fourier transformed using a FFT algorithm.
4. Power spectrum: The resulting spectrum is mapped onto the mel scale but utilising a triangular filter bank.
5. Logarithmic power: The power spectrum is converted to logarithmic scale with respect to the mel frequencies.
6. DCT: The logarithmic mel powers are discrete cosine transformed. The MFCCs is the amplitudes of the output spectrum.

In MATLAB this is done using the toolbox, voicebox[4]. The function for getting the MFCCs is called `melcepst` and a thorough explaination can be found on the voicebox website[5].
In short the function is invoked like this:

```
mel = melcepst(data(:,i), Fs(i), 'M0d', nc, p, n, inc);
```

With **data** being the signal we want to extract the MFCCs from. **Fs** is the sampling frequency of the signal. **'M0d'** is the mode string and the three characters correspond to Hamming window, include 0'th order cepstral coefficient and include delta coefficients. **nc** is the number of cepstral coefficients excluding 0'th coefficient. **p** is the number of filters in the filterbank. **n** is the length of the frame in the samples. and lastly **inc** is the frame increment value. Possible formatting of this block of text
In this project the window size is chosen to be 200 milliseconds. The default value for speaker recognition is 150 milliseconds while speech recognition is typically lower than that. The number of cepstral coefficients was chosen to be 30.

[4]http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html - retrieved 6 june 2015
[5]http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/doc/voicebox/melcepst.html

The mel output from the function is used for features. The resulting matrix is $M \times N$ with M being number of samples of each MFCC and N being number of MFCCs. For the data used in this project his corresponds to a $2175 \times 62$.

## IV. Dimensionality Reduction

e.g. finding projection vectors, choosing number of components, applications.
motivation: data compression speed up learning algorithm

### I. PCA

Principal component analysis is used for reducing the number of dimensions of a feature space. It works by projecting the data in the feature space, down to a fewer dimensional feature space by minimizing the squared projection error. The reduced feature space does not nessecarily share the same features, but new features are found which best retains the variance in the data.
PCA should mainly be used for compressing the data to save memory or reducing running time of learning algorithm. By reducing the amount of features most machine learning algorithms runs faster. PCA can also be used to prevent overfitting, but its usually better to use regularization.
Figure 2 shows a 3 dimensional feature space where all the data, within a small margin, lies in a 2 dimensional plane. PCA is used to find two vectors $u^{(1)}$ and $u^{(2)}$ which spans this 2D plane.

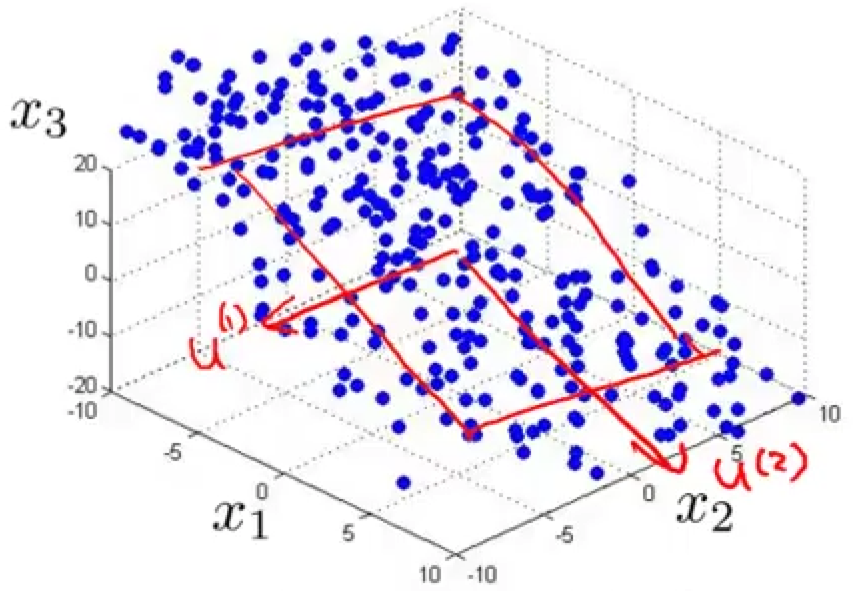

**Figure 2:** *3D to 2D pca illustration*

Preprocessing of the data should be done before doing PCA.
Given the training set:

$$x = \begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix} \tag{1}$$

Ensure that every feature has zero mean by doing mean normalization:

$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)} \tag{2}$$

$$x_j = x_j - \mu_j \tag{3}$$

Feature scaling can also be done if the features have very different value ranges. TODO
After preprocessing the data, we can do PCA on it.
We start by computing the covariance matrix $\Sigma$:

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n} x^{(i)} {x^{(i)}}^T \tag{4}$$

The covariance matrix descripes how the different features relates. When doing feature reduction we want to remove features which has high correlation with other features. An example could be a feature which descripes a length in cm and another feature descriping the same length in inches. These features will have very high correlation and one of them can be removed from the feature space without loosing much information.
Then we compute the eigenvectors of covariance matrix:

$$U = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(n)} \end{bmatrix} \in \mathbb{R}^{n\times n} \tag{5}$$

The eigenvectors will lay in the directions of most variance in the data. This is what is shown on Figure 2. The longer the eigenvector, the more variance it descripes. Therefore we want to keep the longest eigenvectors and remove the shortest eigenvectors.
The eigenvectors are ordered by length in the matrix $U$. The longest is the first.
We select the first $k$ eigenvectors to get the reduced set of eigenvectors:

$$U_{reduce} = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(k)} \end{bmatrix} \tag{6}$$

We can now calculate the new feature vectors:

$$z = U_{reduce}^T x \tag{7}$$

We have now reduced the feature space to a $k$ dimensional feature space. Say we want to retain at least 95% of the variance in the data. We do this by picking the smallest value of $k$ so that:

$$\frac{\sum\limits_{i=1}^{k} S_{ii}}{\sum\limits_{i=1}^{n} S_{ii}} \geq 0.95 \tag{8}$$

The matrix $S$ is found by doing singular value decomposition (SVD). The matrix $S$ has the form:

$$S = \begin{bmatrix} S_{11} & 0 & 0 & 0 & 0 \\ 0 & S_{22} & 0 & 0 & 0 \\ 0 & 0 & S_{33} & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & S_{nn} \end{bmatrix} \tag{9}$$

In our project we use PCA to reduce our feature space from 64 dimensions down to 40 dimensions. We do this to increase the speed of our learning algorithms while still retainging almost all of our data ($\geq$ 99.99%) as seen on Figure 3. (TODO: Add axis labels to figure. x = number of features. y = retained variance.)

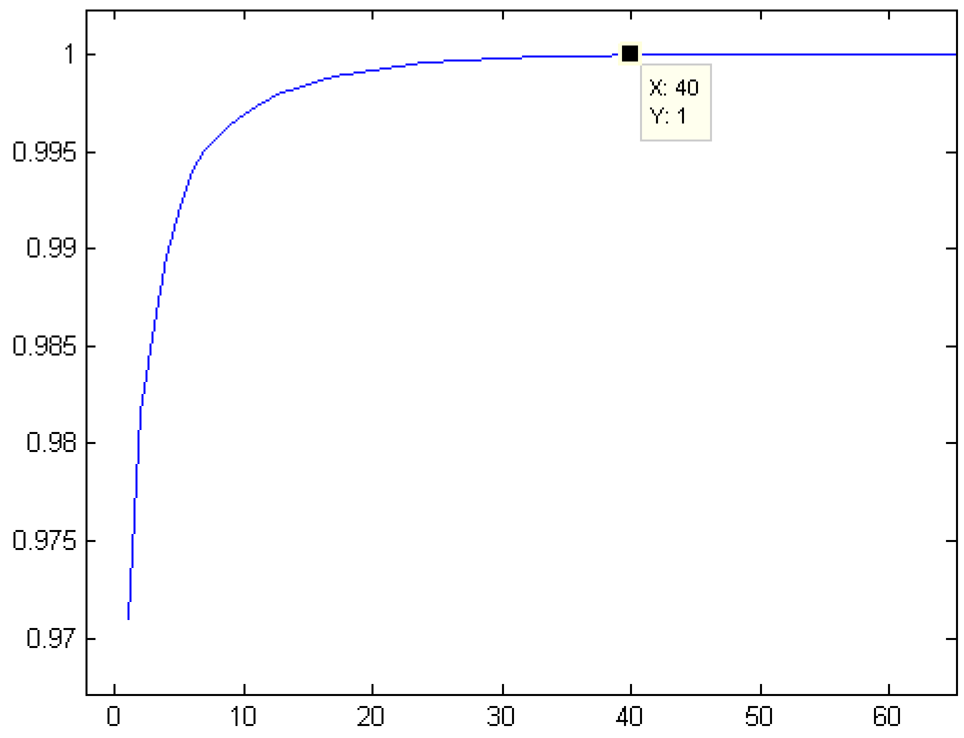

**Figure 3:** *3D to 2D pca illustration*

## II. Fisher

Dimensionality Reduction is another take on linear classification. We use equation 10 and place a threshold on y such that class 1: $y \geqslant -\omega_0$ and class 2 is everything else. This is also said as projecting data down to one dimension.

$$y = \mathbf{w}^T \mathbf{x} \tag{10}$$

This leads to a considerable loss in information and may cause overlapping in data that did not overlap in multidimensional space.
By adjusting the weight vector **w** a solution can be found that minimises overlapping by maximising the distance between classes. Given two classes we have:

$$m_1 = \frac{1}{N_1} \sum_{n=C_1} X_n, m_2 = \frac{1}{N_2} \sum_{n=C_2} X_n \tag{11}$$

With **m** being the mean of the class and `N` being the amount of points in the class. The goal is to maximise the difference $m_2 - m_1$ and this is done by maximising the difference of the projected data as well:

$$m_2 - m_1 = \mathbf{w}^T \times (\mathbf{m}_2 - \mathbf{m}_1) \tag{12}$$

We add the constraint that $\sum_i \omega_i^2 = 1$ in order to avoid large expressions when dealing with the projected data. When solving this we see that there might be an issue with considerable overlap in the projected space.
The fisher method seeks to minimise the class overlap by reducing in class variance. Within-class variance is given by:

$$s_k^2 = \sum_{n=C_1} (y_n - m_k)^2 \tag{13}$$

where $y_n = \mathbf{w}^T \mathbf{x}_n$ and $m_k$ is the difference mentioned earlier. The fisher criterion is given by:

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \tag{14}$$

This can also be written as:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \tag{15}$$

Where $\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T$ is the between-class covariance matrix and
$\mathbf{S}_W = \sum_{n \in C_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n \in C_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T$ is the within-class covariance matrix.
Differentiating and removing the scaling factors we get:

$$\mathbf{w} \propto \mathbf{S}^{-}1_W (\mathbf{m}_2 - \mathbf{m}_1) \tag{16}$$

This is know as Fisher's linear discriminant. This holds for 2 classes but in this project there is 3 classes. A general term for Fisher's discriminant for more than two classes must be considered. Given $\mathbf{y} = \mathbf{W}^T \mathbf{x}$ where $\mathbf{W}$ is the weight vectors $\mathbf{w}_k$. The within-class covariance for K classes is given by:

$$\mathbf{S}_W = \sum_{n=1}^{K} \sum_{n=C_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T \tag{17}$$

With $\mathbf{m}_k = \frac{1}{N_k} \sum_{n=C_k} \mathbf{x}_n$, $N_k$ is the number of patterns in class k. The total covariance matrix given by Duda and Hart(1973) is used to find the between-class covariance matrix.

$$\mathbf{S}_T = \sum_{n=C_k}^{N} (\mathbf{x}_n - \mathbf{m})(\mathbf{x}_n - \mathbf{m})^T \tag{18}$$

where $\mathbf{m}$ is the mean of the total data set. The between class covariance matrix can be found by using $\mathbf{S}_T = \mathbf{S}_W + \mathbf{S}_B$.

$$\mathbf{S}_B = \sum_{n=C_k}^{N} N_k (\mathbf{m}_k - \mathbf{m})(\mathbf{m}_k - \mathbf{m})^T \tag{19}$$

Defining these matrices in the projected space we get:

$$\mathbf{s}_W = \sum_{n=1}^{K} \sum_{n=C_k} (\mathbf{y}_n - \boldsymbol{\mu}_k)(\mathbf{y}_n - \boldsymbol{\mu}_k)^T \tag{20}$$

$$\mathbf{s}_B = \sum_{n=C_k}^{N} N_k (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T \tag{21}$$

where $\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=C_k} \mathbf{y}_n$ and $\boldsymbol{\mu} = \frac{1}{N_k} \sum_{k=1}^{K} N_k \boldsymbol{\mu}_k$.

The cost function is then defined as:

$$J(\mathbf{W}) = Tr\mathbf{s}^{-}1_W\mathbf{s}_B \tag{22}$$

The resulting dimensions is K - 1 where K is the number of classes. In this project the dimensions of the fisher linear discriminant is $3 - 1 = 2$. Analysing these features does not provide usable classification for this project.

## V. Classifiers

Classifiers were first know from the world of linear regression. The classifiers found in this section are featured in the non-linear signal processing and pattern recognition course. The section seeks to explain the basis of each of the classifiers along with how we have used them in our project. Intermediate results can be found in the section about the classifiers while the comparison between classifiers can be found in the Results section.

### I. Linear Classifier

The goal of linear classification is to take an input vector with multiple x values and assign it to one of multiple classes K. This can be done with one or more linear decision boundaries. The first way to classify is called the one-vs-one linear classifier. This works for 2 classes as seen in figure 4. If multiple clusters of x belonging to more than 2 classes are present we get ambiguous regions as one class might appear to be two different classes. An example of this can be seen in figure 5.

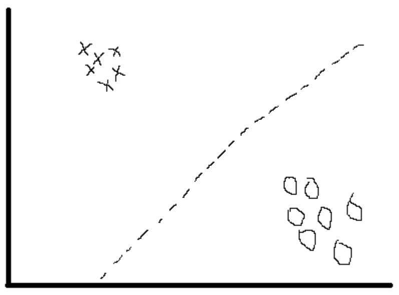

**Figure 4:** *One-vs-one linear classifier for 2 classes*

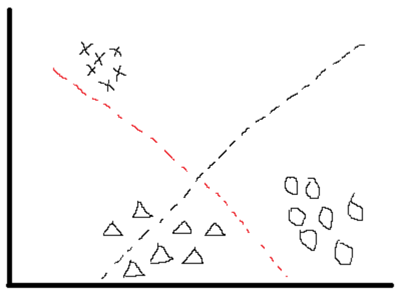

**Figure 5:** *One-vs-one linear classifier for 3 classes*

Another way to classify the 3 classes seen in figure 5 could be to utilise 1-of-k classification. This can be seen in figure 6. The 1-of-k classifier has no ambiguity in this case.

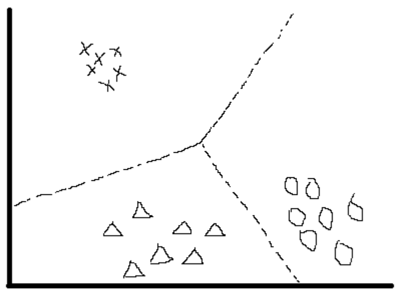

**Figure 6:** *1-of-k linear classifier for 3 classes*

In math terms the one-vs-one can be written as:

$$y(\mathbf{x}) = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}} \tag{23}$$

This is because we can consider the output y to be a weighted sum of the inputs. The error function can be defined as:

$$E(w) = \Sigma_n (\hat{y}(w, x_n) - y_n)^2 \tag{24}$$

Where $\hat{y}$ is the estimated $y$ value and $y_n$ is the true $y$ value.

If we look at a case with more than two classes, the linear classifier is prone to ambiguity. We know that the ambiguity issue can be avoid by using the form:

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + \omega_{k0} \tag{25}$$

and choosing the value of x to be a part of class k if $y_k(\mathbf{x}) > y_m(\mathbf{x})$ for all $m \neq k$. This leads to decision boundaries corresponding to the 1-of-k classifier where the decision boundaries join together in the middle corresponding to the image in figure 6.

**Training:**
Training the one-of-k function requires the use of two vectors in matlab: $t \& Z$. t is vector of the correct classes while Z is a vector containing our features. In order to train the one-of-k classifier we use the following equation:

$$w^* = (Z^T Z)^{-1} Z^T t \tag{26}$$

This results in the estimated weights for the classifier. To classify the data we use the cost function described earlier in equation 23.

In order to observe the boundaries in the project, the data must be 2 or 3 dimensional. This will require either the use of the PCA or fisher reduction methods explained in early sections. This leads to the image seen in figure 7.

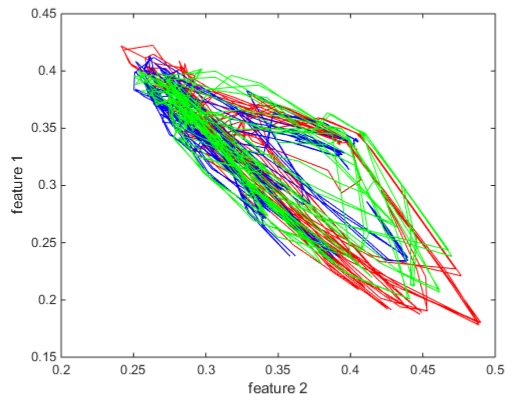

**Figure 7:** *2 dimensional 1-of-k linear classifier for 3 classes of speech*

This does not provide a usable visual representation of the classifier. The choice was made to keep the data in the higher dimensions. The output from the cost function provides a sample and the values representing the three classes:

```
0.5333
0.2506
0.2160
```

The cost function classifies the first sample to belong to class 1 as an example.
**Intermediate results:**
The test data was split into 3 sections and run through the cost function. This resulted in three plots as can be seen in figure 8, 9 & 10. The classes are coloured: Class 1 = Red, Class 2 = Blue, Class 3 = Green.

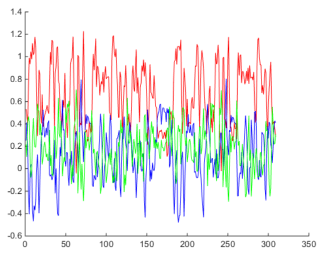

**Figure 8:** *Output from first 1/3 of the data*

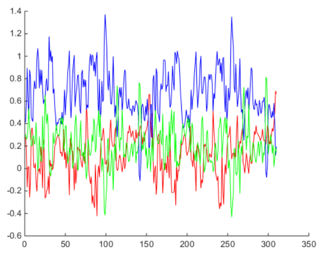

**Figure 9:** *Output from middle 1/3 of the data*

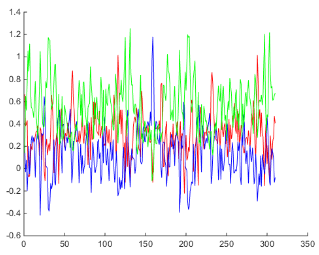

**Figure 10:** *Output from last 1/3 of the data*

When evaluating the peak values of the three plots, it can be observed that the first 1/3 of the data belongs to class 1, the middle 1/3 of the data belongs to class 2 and the rest of the data belongs to class 3.

## II. Probabilistic models

In many cases the features follows a certain distribution. By using the information in the distribution it is possible to filter out outliers and determine how likely it is that the point belongs to a certain class. This can be very powerful since we not blindly put a sample in a class but also get information about the likelihood. This of course demands that a qualified guess of the distribution can be made. By using the probability for a given sample, given a certain class $P(x|C)$. Iteration over the the different classes, it is possible to decide were to classify the sample, and how certain we are of this decision. This is illustrated in figure 11

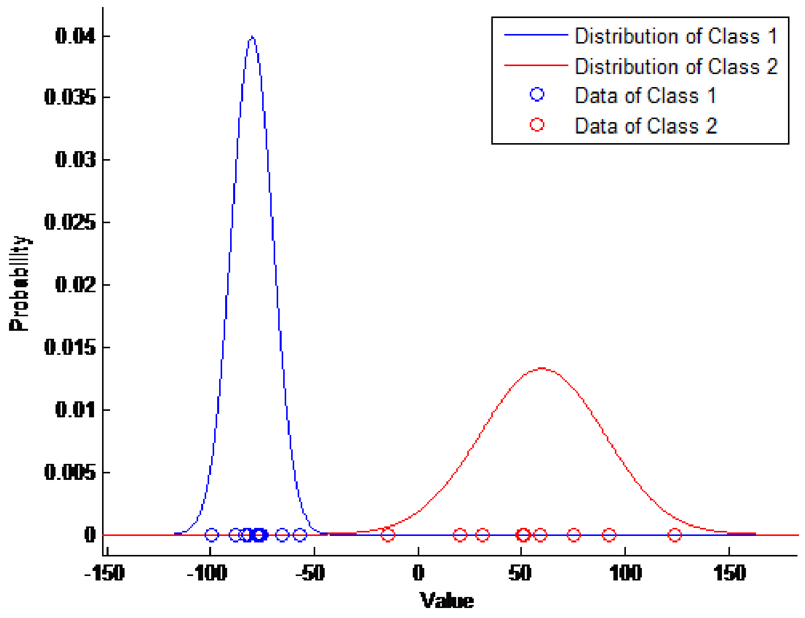

**Figure 11:** *Distributions of 2 Classes*

## II.1 Discriminative Model

Instead of asking what the probability of the sample, given the class is $P(x|C)$, the conditional probability can be used. That is the probability of a class, given a sample $P(C|x)$. This can be found using Bayes rule:

$$P(C|x) = \frac{P(x|C)P(C)}{P(x)} \tag{27}$$

This kind of model is called a discriminative model, and is used for modelling the dependence of an unobserved class $C$ on an observed variable $x$. This will for the Gaussian distribution be a sigmoid function. Using this model to classify it is no longer able to tell about the probability of a sample not being in any class, but on the same time also simplifies the classifier a lot. Compared to the linear classifier the discriminative classifier are able to create a mouth sharper decision bound. If the sharper decision bound is the goal then a easer approach is to estimate the optimal sigmoid for separation of classes directly. This can be done by optimizing a soft-max function to separate the classes. The soft-max function is expressed as:

$$y_k(\mathbf{w}_k, \mathbf{x}) = \frac{e^{\mathbf{w}_k \mathbf{x}}}{\sum\limits_{k=1}^{K} e^{\mathbf{w}_k \mathbf{x}}} \tag{28}$$

Comparing this to the previous probabilist function we can assume that:

$$P(t|\mathbf{w}, \mathbf{x}_n) = p_n^t(1-p_n)^{1-t}, t \in [0, 1] \tag{29}$$

Here we see how likely it is that the class vector t, is correct given data point x, and some weights w in the soft-max. Where t is the class vector, that indicates which class the data point x is part of. The $p_n$ is given by:

$$p_n = P(C|\mathbf{w}, \mathbf{x}_n) = y(\mathbf{w}, \mathbf{x}_n) \tag{30}$$

The challenge is now to find the optimal weights $w_k$, for each class, to create the best classifier. This can be done by combining the two equations 28, 29 to create a non linear optimisation problem.

$$L(w) = \log \prod_{i=1}^{N} y(\mathbf{w}, \mathbf{x}_i)^{\mathbf{t}_i}(1 - y(\mathbf{w}, \mathbf{x}_i))^{1-\mathbf{t}_i} = \sum_{i=1}^{N} t_i \log y(\mathbf{w}, \mathbf{x}_i) + (1 - \mathbf{t}_i) \log(1 - y(\mathbf{w}, \mathbf{x}_i)) \tag{31}$$

This can be solved by many different optimizations strategies. By optimizing this for each class we will find the optimal weights for the soft-max class separator in equation 28.
In the speaker recognition case a soft-max function has been used to create a discriminative model, that are able to separate the classes as described above. A full probabilistic model using the $P(x|C)$ approach, has also been attempted by using a Gaussian mixture model. This is described in the section: II.6. In this analysis it is assumed that the data is Gaussian distributed. This assumption is made by looking at the histogram of the features. This is shown on figure 12.

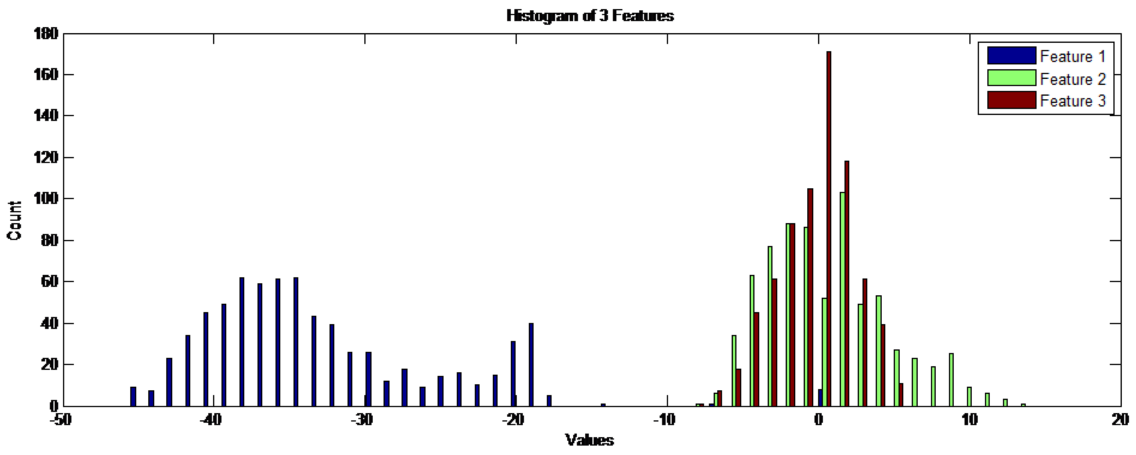

**Figure 12:** *Histogram of 3 features*

By running the optimizing the soft-max function a model is found that are capable of separating the test set in the three classes, whit a 21.58 % error rate. The results can be seen in table 1.

| Total Error | 21.58 % |
|---|---|
| Nicolai Error | 24.27 % |
| Rasmus Error | 10.68 % |
| Rune Error | 29.77 % |

**Table 1:** *Discriminative Model Results*

The results indicate that Rune is the one there is hardest to classify. Even though some samples are classified wrong, the majority of samples is correct classified. This makes this method valid for speaker recognition.

have awesome glud billed .

## II.2 Gaussian Mixture Models

One of the most common distributions is the Gaussian distribution $\mathcal{N}(\mu, \sigma)$. Sometimes this model is not quiet enough to capture the underlying distribution. In such cases can a mixture of Gaussian distributions be created to fit a underlying distribution. This principal is shown in figure 13.

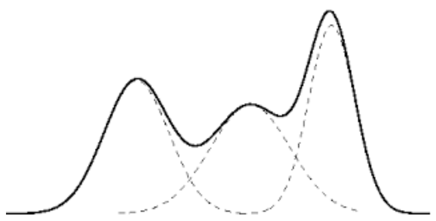

**Figure 13:** *Gaussian Mixture Model Principe*

This method can be generalized up in a multidimensional space, to fit the feature space. In a multidimensional model we introduce the covariance matrix $\Sigma$ that like $\sigma$ gives the deviation of a Gaussian model. The Gaussian mixture model can therefore be described as:

$$P(x) = \sum_{k=1}^{K} \Pi_k \mathcal{N}(x|\mu_k, \sigma_k) \tag{32}$$

Where $\Pi_k$ is a weight called the mixture parameter, that is used to differentiate the individual Gaussian distributions in the model. knowing this we can also describe the mixture model from equation 32 as:

$$P(x) = \sum_{k=1}^{K} P(k)P(x|k) \tag{33}$$

In order to use a the mixture model, the different distributions must first be found. This topic will be discussed in section II.3 and II.4

## II.3 k-means Algorithm

One way of finding groupings of data, in the feature space, is by using the k-means algorithms. This algorithm tries to split the data in k groups. This is done by following 4 three step iterating proses.

1. Initialize means: Select k random mean values in the feature set.
2. Assign responsibility: For each point, find the closest mean points and make it responsible for that point.

3. Calculate new mean: Move each mean point to the mean value of the cluster of points the mean is responsible for.

4. Continue whit step 2, till no change in points.

On figure 14 is illustrated how the k-means in 6 iterations split the data up in 3 clusters.

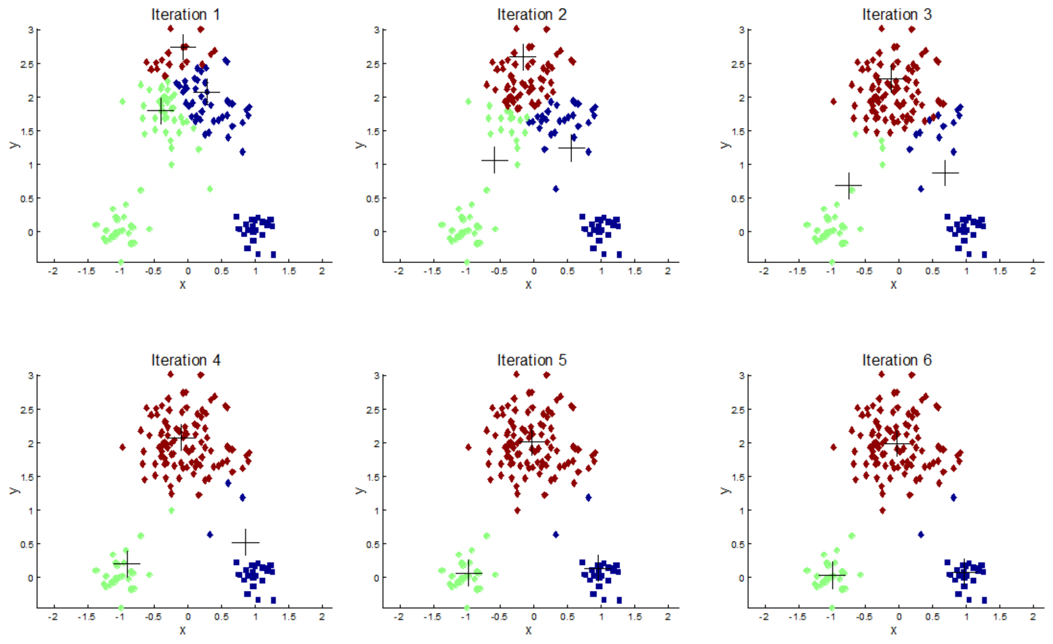

**Figure 14:** *K-means used on a data set*

The result of this proses will differ depending on the initial mean guesses. This is especially the case is there is no natural groupings in the dataset. In this case the algorithm will still split up the data in k clusters there are side by side. This is illustrated on figure 15, where the k-means algorithm tries to cluster points distributed uniformly on a circle, in 7 clusters.

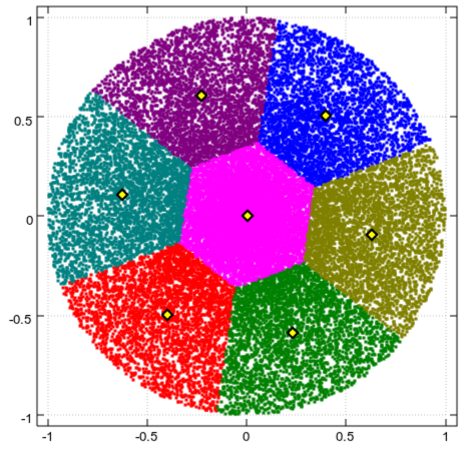

**Figure 15:** *K-means used on points distributed uniformly on a circle*

### II.4 EM Algorithm For Gaussian Mixture Models

As discussed in section II.2 it is important to have a method of finding the best Gaussian distributions in the data set to create a good mixture model. The best Gaussian Mixture Models that fits the data can be described as optimising equation 34.

$$L(x) = \sum_{n=1}^{N} \log \sum_{k}^{K} P(k) P(\mathbf{x}_n | k) \tag{34}$$

To find the maximum of equation 34, we find the differentiated and set et equal zero.

$$\frac{\partial L(\mathbf{x})}{\partial \boldsymbol{\mu}_k} = 0 \tag{35}$$

From equation 32 and 35 the following optimisations equations can be found:

$$N_k = \sum_{k}^{K} P(k|\mathbf{x}_n) \tag{36}$$

$$P(k) = \Pi_k = \frac{N_k}{N} \tag{37}$$

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n}^{N} P(k|\mathbf{x}_n)(\mathbf{x}_n - \boldsymbol{\mu}_n)(\mathbf{x}_n - \boldsymbol{\mu}_k)^\intercal \tag{38}$$

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n}^{N} P(k|\boldsymbol{x}_n)\boldsymbol{x}_n \tag{39}$$

The variable $N_k$ can be seen as the effective number of samples in a cluster, and $N$ is the total number of samples. The $P(k|x_n)$ term can be seen as the responsibility a center point has for that point. Unlike the K-means algorithm the all center points are responsible for all points, but the level of responsibility is different from center point to center point. To find the optimal center points $\mu_k$, and there deviations $\Sigma_k$ the EM algorithm can be used. This is based on a the idea of a E-step, called the estimation step, and a M-step called the maximisation step. Much like the k-means does this also have 4 iterative steps.

1. Initialize Parameters: Select k random values for $\mu_k$ , $\Sigma_k$ , $\Pi_k$
2. E-Step: Update the responsibility by using Bayes' theorem stating:

$$P(k|x) = \frac{P(x|k)P(k)}{P(x)} = \frac{P(x|k)P(k)}{\sum\limits_{k}^{K} P(x|k)P(k)} \tag{40}$$

3. M-Step: Maximize the Gaussian mixture model parameters by using equations 36, 37, 38 and 39.
4. Continue whit step 2, till no change in Gaussian mixture model parameters.

Like the k-means algorithm does this algorithm also map a predetermined number of Gaussian distributions to the dataset. The optimal amount must be found though exploration. On figure 16 we see how the EM algorithm has been used to map 3 Gaussian mixture model to a dataset.

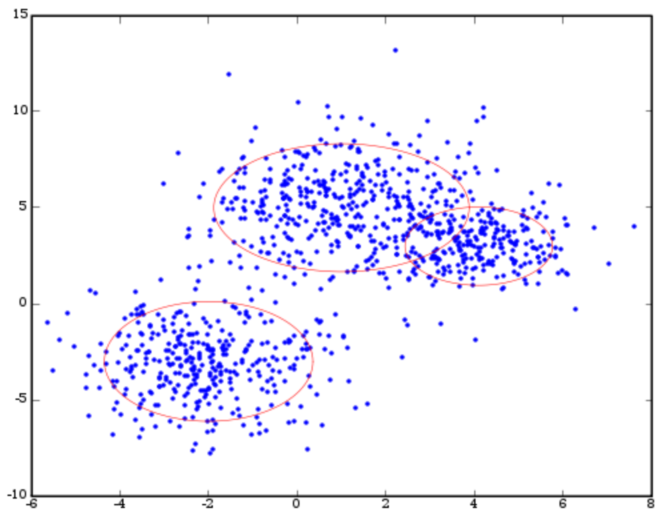

**Figure 16:** *EM Algorithm Used On Data To Create GMM*

Like for the k-means the initial start estimates for the mean value is important for a good result, and a fast execution time. A normal approach is to use the k-means algorithm first, and use the final means as the start estimate in the em algorithm. In order to reduce the training time, the model is sometimes reduced in complicity. Instead of a full Gaussian distribution a spherical covariance or a completely diagonal covariance is used.

### II.5 Unsupervised Gaussian Mixture Model

Sometimes the training data is unlabelled, which mean that the classes is unknown. In such data there can still be some underlying structures, that can be used to classify the data. The goal of unsupervised learning is to discover these structures, and create the classes from them.

One approach of this is to select a number of desired classes, and try finding clusters in the data to fit to the classes. This can be done whit the k-means as shown in figure 14. where 3 classes is found to fit the data. Likewise can the Gaussian mixture model also be used to find distributions that describes the classes. This can be seen in figure 16 where 3 Gaussian distributions is fit to the unlabelled data.

In the speech recognition case we have investigated if it is possible to use the Gaussian mixture model as a unsupervised method to find the 3 persons: Nicolai, Rasmus and Rune. For this to work the features must be placed in separated clusters corresponding to the persons.

When evaluating this we see that the classes found does not correspond to the 3 know classes. The when looking at the errors on table 2, it is seen how the error for every one over 50

| Total Error | 65.24 % |
|---|---|
| Nicolai Error | 53.10 % |
| Rasmus Error | 78.34 % |
| Rune Error | 64.27 % |

**Table 2:** *Unsupervised GMM Results*

This means that this method is not able to automatically split the data up in the desired classes.

## II.6 Supervised Gaussian mixture model

What if we used the same method to train the data, but in a supervised manner. A other method is to train a Gaussian mixture model for each class, and use this to classify. A Gaussian mixture model was trained to each voices. To find the optimal amount of Gaussian distributions was a experiment where we started at one and ended whit 10.

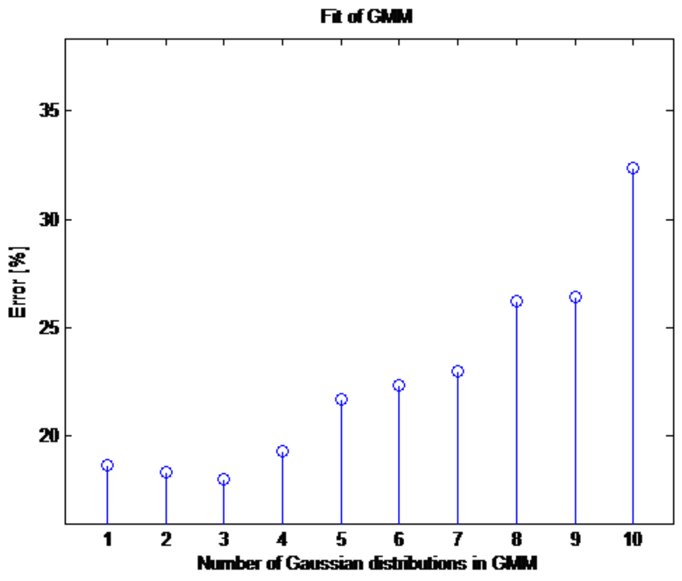

**Figure 17:** *GMM fit in relation to number of Gaussian distributions*

As seen on figure 17, three Gaussian distributions in each GMM is the one that gives the lowest error. To ensure that 18% is the best we can achieve whit this model, the model is retrained whit random initializations. The best fit found whit this configuration can be seen in table 3.

| Total Error | 15.64 % |
|---|---|
| Nicolai Error | 9.06 % |
| Rasmus Error | 21.68 % |
| Rune Error | 16.18 % |

**Table 3:** *Supervised GMM Results*

Here it is seen how this model is relative good at separating the data in the the three classes. This model have the most trouble finding Rasmus, this is noticeable different from the other models where Rune is the hardest to recognize.

## III. Artificial Neural Network

### III.1 Introduction

Artificial neural networks are an attempt at creating a modelling how a brain works. A network consists of neurons connected together. A neuron takes zero or more inputes, does some computation and gives one input as seen on figure 18.

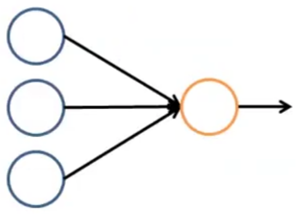

**Figure 18:** *A single neuron. The 3 blue circles are input units. The orange circle is the neuron.*

Neurons can be combined into networks as seen on figure 19. How the neurons are connected is called the network architecture. The first layer is called the input layer. The last layer is called the output layer. All other layers are called hidden layers. The nodes in the network are called units. The network on figure 19 has 3 input units and 1 output unit which in machine learning terms means it takes 3 features and does binary classification. To do multi-class classification, the output layer shall contain one output unit per class.

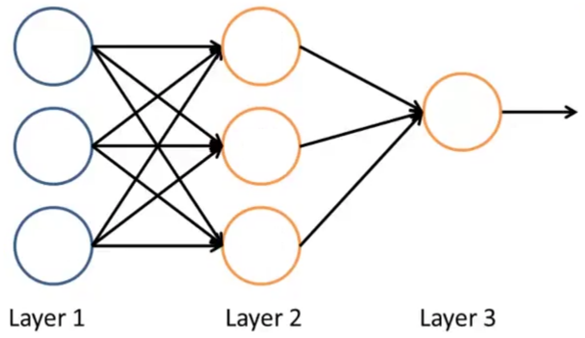

**Figure 19:** *An artificial neural network with 3 layers.*

Neural networks can be thought of as an extension of logistic regression. In logistic regression we would only have layer 2 and 3, where layer 2 would contain the features. In neural networks we can add more layers, where each new layer maps the features of the previous layer to a set of new features. This gives more flexibility and expressive power in the model.

### III.2 Mathematical representation

$a_i^{(j)}$ is the activation of unit $i$ in layer $j$, which uses the Sigmoid activation function: (TODO: show a plot. What does activation mean?)

$$a(x) = \frac{1}{1 + e^{-\Theta^T x}} \tag{41}$$

$\Theta^{(j)}$ is a matrix of weights, controlling function mapping from layer $j$ to layer $j+1$. If the network has $s_j$ units in layer $j$ and $s_{j+1}$ units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$. Each layer also contains a bias unit $a_0^{(j)}$. The representation is illustrated on figure 20.
(TODO: Show how $\Theta^{(j)}$ is indexed on the figure.)

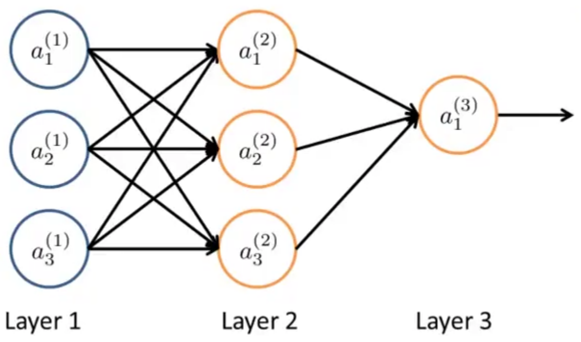

**Figure 20:** *An artificial neural network with 3 layers.*

The activation of a layer is defined as:

$$a^{(j+1)} = g(\Theta^{(j)} a^{(j)}) \tag{42}$$

The activation of a layer is therefore depent on the activation of the previous layer. So you have to start by activating the first layer and then propagate through the layers, hence the name of the algorithm; "forward propagation algorithm".
We can now compute the layer activations one at a time to get the output of the network on figure 20:

$$a^{(2)} = g(\Theta^{(1)} a^{(1)}) \tag{43}$$

$$a^{(3)} = g(\Theta^{(2)} a^{(2)}) = g(\Theta^{(2)} g(\Theta^{(1)} a^{(1)})) \tag{44}$$

The activation of the output layer is also called:

$$h_\Theta(x) = a^{(3)} \tag{45}$$

### III.3 Training the network

Given $m$ training examples $\left\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\right\}$. Training the neural network is an optimization problem, where a cost function $J(\Theta)$ should be minimized with respect to the weights $\Theta$:

$$\min_\Theta J(\Theta) \tag{46}$$

We can minimize the cost function using an optimization algorithm like gradient descent. For such an optimization algorithm, we also need the partial derivatives of the cost funtion:

$$\frac{\delta}{\delta\Theta_{ij}^{(l)}} J(\Theta) \tag{47}$$

The cost function $J$ is defined as:

$$\begin{aligned} J(\Theta) = &-\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) log(1-(h_\Theta(x^{(i)}))_k)\right] \\ &+\frac{\lambda}{2m}\sum_{k=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2 \end{aligned} \tag{48}$$

Where $L$ is the total number of layers, $s_l$ is the number of units in layer $l$, $K$ is the number of output units and $m$ is the number of training examples.
The second term of the cost function is the regularization term.
(TODO: Explain the intuition behind the cost function.)
To find the partial derivatives, we can use the backpropagation algorithm: (TODO: Help please)
The error term of the output layer is defined as:

$$\delta^{(L)} = a^{(L)} - y \tag{49}$$

The error term of the hidden layers is defined as:

$$\delta^{(l)} = (\Theta^{(l)})^T \delta^{(l+1)} .* a^{(l)} .* (1 - a^{(l)}) \tag{50}$$

There is no error term of the input layer.
The partial derivative can then be found as:

$$\frac{\delta}{\delta\Theta_{ij}^{(l)}} J(\Theta) = \delta \tag{51}$$

(TODO: Show figure of how back propagation works.)
(TODO: How do we use it?)

## IV. Sequential Models

The classifiers presented until this point have been stateless. This means that each point is being classified solely on its values, and not in the context of samples. But sometimes information about how the features change over time can be used to determine the class. An example of this is in word recognition, where the order the letters come in is important for recognizing a word. If we look at the following two words:

- "Cow"
- "Coooww"

It is easy to see that it is the same word, some of the letters is just repeated in the second case. This is often the case when trying to recognize words in speech, due to the variance in speed from person to person. To model this we could look for the probability of:

$$P(x_1, x_2, ..., x_5) = P(x_5|x_4, x_3, x_2, x_1)P(x_4|x_3, x_2, x_1)P(x_3|x_2, x_1)P(x_2|x_1)P(x_1) \tag{52}$$

Which simply states that the probability of a given sequence, is the probability of the first letter multiplied by the probability of the next letter, given the first, and so on. To model this kind of behavior a Markov model can be used.

## IV.1 Markov Model

The Markov model is a way of modelling a sequential dependency in the data. Much like before does the data need to be spilt into classes, called states. The The Markov model does not deal whit splitting the data into states, but rather in what order the data transitions between states.

In the Markov model it is assumed that the state only is determined by the last sample. Mathematically described as shown in equation 53.

$$P(x|x_{n-1}, x_{n-2}, ..., x_1) = P(x|x_{n-1}) \tag{53}$$

This assumption simplifies equation 52, so we know have:

$$P(x_1, x_2, ..., x_n) = P(x_1) \prod_{n=2}^{N} P(x_n|x_{n-1}) \tag{54}$$

This is also called a Markov chain or a first order Markov model, since it is only depended on the last sample. Variations of the model does exists that takes more than just the last sample in consideration, but will not be further discussed in this report.

A simple Markov model is shown in figure 21. This is a illustration of the previous "Cow" example, where each letter is a state. Here we assume that we always start in state "C". From here we have 95% chance of moving to the "O" state and 5% chance of staying in the "O" state.

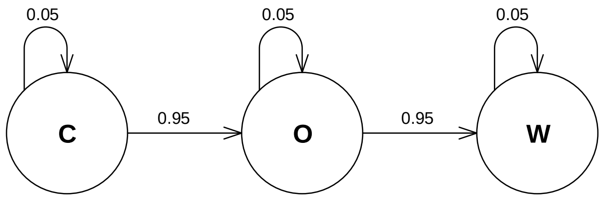

**Figure 21:** *Markov model for the word "COW"*

In this model there is 0% chance of moving from state "C" to "W". It is only possible to go to the same state, or change to the next in the letter series. This kind of model is called a Left-to-right model. This model is often used in this kind off application, but for other applications a other model where it is possible to jump between many states could be relevant. The states transaction is often given in a transaction matrix:

$$A_{n,n-1} = \begin{bmatrix} 0.05 & 0.95 & 0 \\ 0 & 0.05 & 0.95 \\ 0 & 0 & 1 \end{bmatrix} \tag{55}$$

In equation 55 the transition matrix is shown for the word "COW" illustrated by the transition graph in figure 21. The matrix is an $M \times M$ matrix, where $M$ is the number of states. Each row and column is symbolizing the states, and the cells contains the transition probability from a row element to a column element.

## IV.2 Hidden Markov Model

A hidden Markov model is a Markov model in which the system being modelled is assumed to be a Markov process with unobserved states, also known as hidden states. It also have a set of desired output states. Like for the normal Markov model there is a transition between the hidden states given by a transition matrix $A$. For each hidden state there is also a probability of a given output state this is called the emission probability.

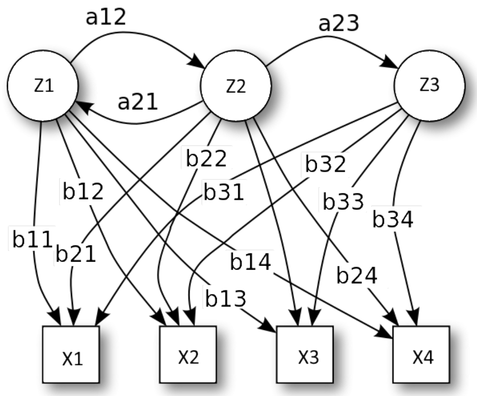

**Figure 22:** *Hidden Markov Model*

This is illustrated on figure 22, where x is the known output states. The z is the hidden input states and a is the transaction parameters and b is the emission parameters. the probability for a given hidden sequence and output can be described as:

$$P(x_1, ..., x_n, z_1, ..., z_n) = P(z_1) \prod_{n=2}^{N} P(z_n|z_{n-1}) \prod_{n=1}^{N} P(z_n|z_n) \tag{56}$$

For many applications we do not know the hidden state of the data, but only the desired output state. It is therefore desired to make a training session that finds the optimal hidden states, and transition and emission parameters that gives us the best fit to the desired output states.
For this the EM algorithm can be used. To do the estimation step a function called the $\gamma$-function ans the $\xi$-function is used:

$$\gamma(z_n) = p(z_n|x_1, ..., x_n) \tag{57}$$

$$\xi(z_n, z_{n-1}) = P(z_n, z_{n-1}|x_1, ..., x_n) \tag{58}$$

Using the this equation it is possible to create the EM steps to find the optimal parameters.

### IV.3 Forward - Backward Algorithm

When evaluating how likely it is that some given data $x_1, x_2, ...x_n$ is fitting a given hidden Markov model, the probability can be described as such:

$$P(x_1, x_2, ..., x_N) = \sum_{z_1} \sum_{z_2} ... \sum_{z_M} P(x_1, ..., x_N, z_1, ..., z_N) \tag{59}$$

This can be a extensive calculation, and can be reduced much in complexity by using the Forward - Backward algorithm. The algorithm works by spiting the calculation up in a forward $\alpha$-step, and a backward $\beta$-step. Looking at the alpha step:

$$\alpha(z_n) = P(x_1, ..., x_n, z_n) = \sum_{z_{n-1}} \alpha(z_{n-1}) P(x_n | z_n) \tag{60}$$

Using the relations in equation 60, we can find the probability of $P(x_1, x_2, ..., x_N)$ by:

$$P(x_1, x_2, ..., x_N) = \alpha(z_N) \tag{61}$$

In much the same way can the $\beta$-step we given by:

$$\beta(z_n) = P(x_{n+1}, ..., x_N, z_n) = \sum_{z_{n+1}} \beta(z_{n+1}) P(x_{n+1} | z_{n+1}) P(z_{n+1} | z_n) \tag{62}$$

This can be used to find the probability of a hidden state given a output sequence.

$$P(z_n | x_1, ..., x_N) = \alpha(z_n) \beta(z_n) \tag{63}$$

### IV.4 Hidden Markov Model In Speaker Recognition

In the speaker recognition case we did not uses the Markov model since there is no sequential dependency in the data, in the same way as for the word example.
Some different approaches was tried to fit a hidden Markov model to the data anyway, but no usable output was produced. This is due to the fact that there is no natural clusters in the data, as discussed in chapter II.5. This means that the created hidden states is very random, and the jumps between them is therefore irrelevant.

## V. Support Vector Machines

The support vector machine is a supervised learning algorithm. Compared to neural network, it sometimes gives a cleaner and more powerful way of learning complex non-linear functions. (TODO: What does more powerful mean?)

### V.1 Linear

When doing the linear classifier, SVM will try to find the line that seperates the data with the highest margin as seen on figure 23.

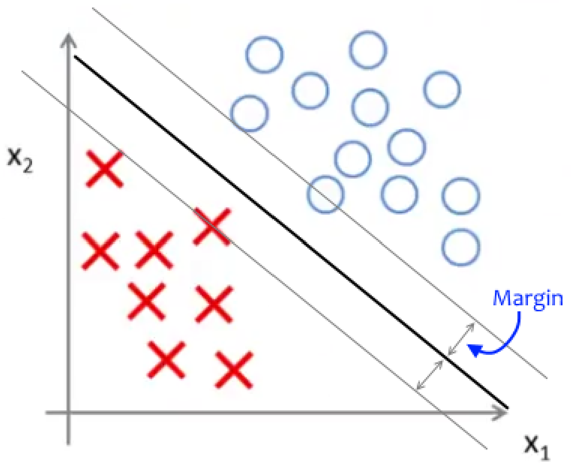

**Figure 23:** *Large margin classification.*

By solving the optimization problem defined as equation 64, we find the parameters for the linear SVM classifier.

$$\min_{\theta} C \sum_{i=1}^{m} \left[ y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2 \tag{64}$$

We must choose the value of the constant $C$. (TODO: what is its effect? Some regularization thing.) Figure 24 shows the plot of $cost_0$ and $cost_1$.

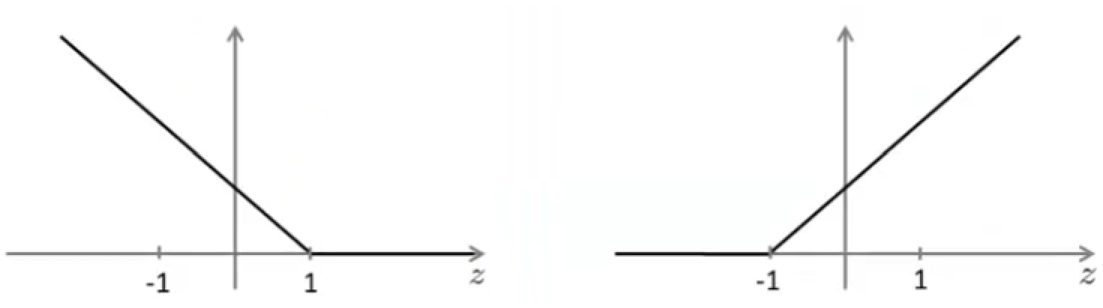

**Figure 24:** *Left:* $cost_1(\theta^T x^{(i)})$. *Right:* $cost_0(\theta^T x^{(i)})$

Hypothesis: (TODO: Is this correct? shouldn't it be 1 if $\theta^T x \geq 1$ and 0 if $\theta^T x \leq -1$)

$$h = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{65}$$

### V.2 Non-linear

Finds the kernels that seperates the data best. gaussian kernels.

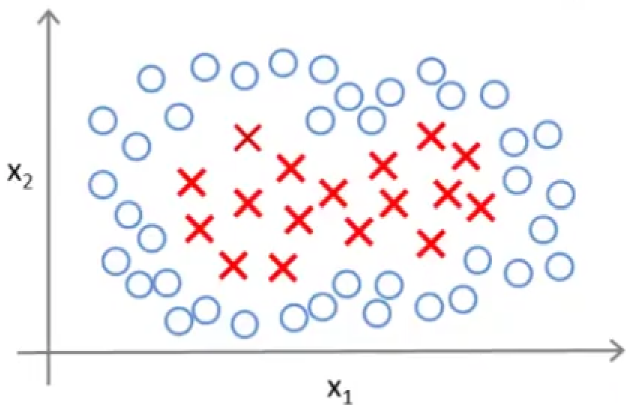

**Figure 25:** *Example of problem that requires non-linear classification.*

The gaussian kernel function:

$$f_i = similarity(x, l^{(i)}) = exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right) \tag{66}$$

At each training example we will put a landmark:

$$\begin{aligned} &\text{Given } (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)}), \\ &\text{choose } l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)} \end{aligned} \tag{67}$$

We convert the training data into new feature vectors. For training example $(x^{(i)}, y^{(i)})$:

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix} \tag{68}$$

where $f_0^{(i)} = 1$.
The minimization problem now looks like:

$$\min_\theta C \sum_{i=1}^{m} \left[ y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{m} \theta_j^2 \tag{69}$$

A large C gives lower bias and high variance, which means it uses a smaller amount of regularization and is more prone to overfitting.
A small C gives higher bias and low variance, which means it uses a higher amount of regularization and is more prone to underfitting.
A large $\sigma^2$ gives higher bias and lower variance, which means that features $f_i$ varies more smoothly.
A small $\sigma^2$ gives lower bias and higher variance, which means that features $f_i$ varies less smoothly.
(TODO: How do we use it?)

## VI. Results

Compare all the methods in a table in order to show the performance.

## VII. Discussion

### I. Subsection One

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

### II. Subsection Two

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

## VIII. Conclusion

## List of Corrections