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

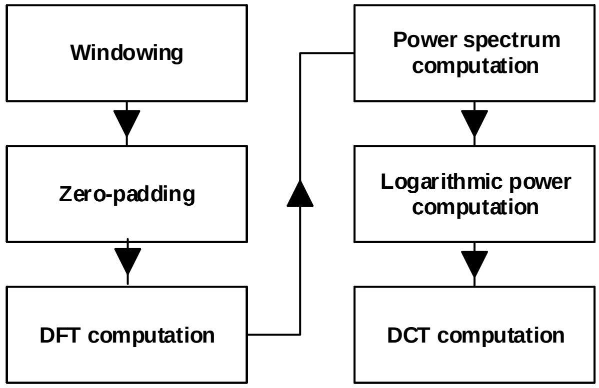

Figure 1: *MFCC steps*

The figure has been derived from the article [1]. The steps can be described as follows:

1. Windowing
2. Zero-padding
3. DFT
4. Power spectrum
5. Logarithmic power
6. DCT

Math

How we use it

IV. Features

Size and number of features and stuff.

V. 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.

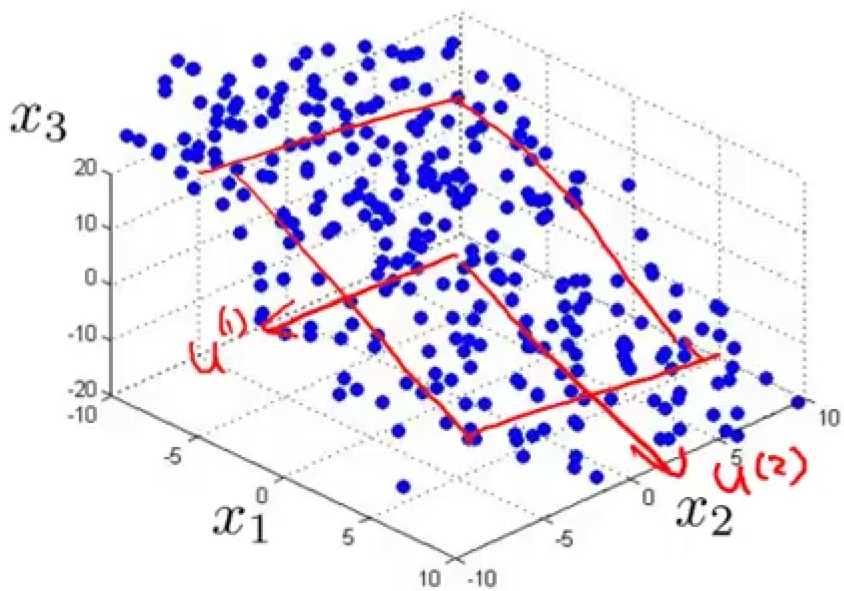

Figure 2: *3D to 2D pca illustration*

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 Σ:

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

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:

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

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.

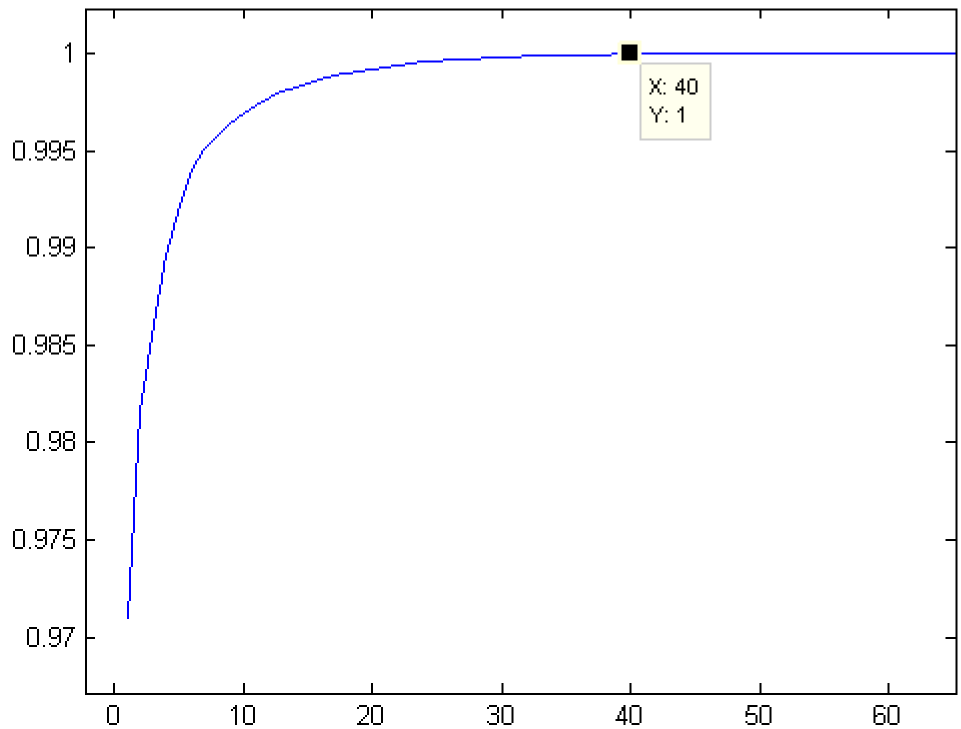

Figure 3: *3D to 2D pca illustration*

II. Fisher

Introtext

Math

How we use it or why we don't use it

Intermediate result

VI. 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.

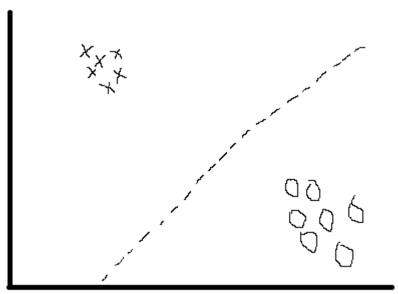

Figure 4: *One-vs-one linear classifier for 2 classes*

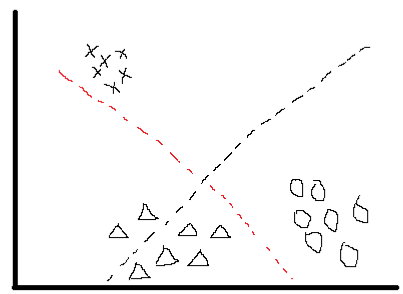

Figure 5: *One-vs-one linear classifier for 3 classes*

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.

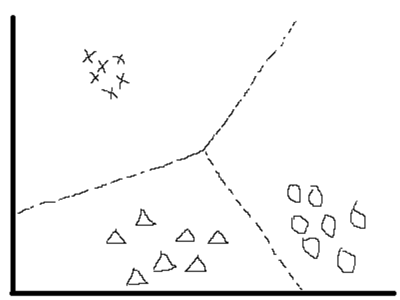

Figure 6: *1-of-k linear classifier for 3 classes*

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

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

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

$$E(w) = \Sigma_n (\hat{y}(w, x_n) - y_n)^2 \tag{11}$$

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:

$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + \omega_{k0} \tag{12}$$

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:

$$w^* = (Z^T Z)^{-1} Z^T t \tag{13}$$

This results in the estimated weights for the classifier. To classify the data we use the cost function described earlier in equation 10.

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.

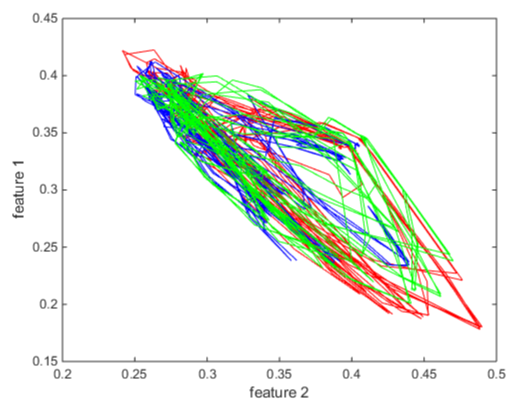

Figure 7: *2 dimensional 1-of-k linear classifier for 3 classes of speech*

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.

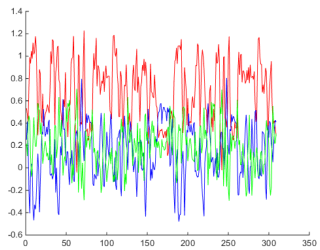

Figure 8: *Output from first 1/3 of the data*

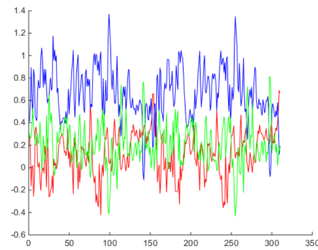

Figure 9: *Output from middle 1/3 of the data*

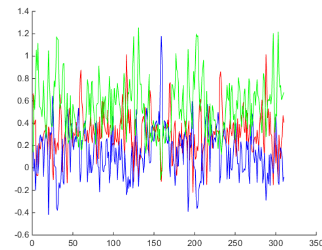

Figure 10: *Output from last 1/3 of the data*

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. Probability Classifier

e.g. maximum likelihood, training/testing and generative vs. discriminative models.

Introtext

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 is a part of our 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 we are able to give a qualified guess of the distribution to use. 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 (fix me).

show figure 1

By using the probability for a given sample if a certain class is assumed $P(x|C)$. Iteration over the the different classes we can get the probability of the sample given these classes. This information can be used to determine what class and how certain we are of this decision. This is illustrated in figure X. (fix me)

figure 2

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

$$P(C|x) = \frac{P(x|C)P(C)}{P(x)} \tag{14}$$

This will for the Gaussian distribution something like 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 this probabilistic 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 softmax-function to separate the classes. The softmax-function is expressed as:

$$Y_k(w_k, x) = \frac{e^{w_k x}}{\sum\limits_{k=1}^{K} e^{w_k x}} \tag{15}$$

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

$$P(t|w, x) = p_n^t (1 - p_n)^{1-t}, t \in [0, 1] \tag{16}$$

Here we see how lilly 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:

$$p_n = P(C|w, x) = y(w, x) \tag{17}$$

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 15, 16 to create a non linear optimisation problem.

$$L(w) = \log \prod_{i=1}^{N} y(w, x_i)^{t_i} (1 - y(w, x_i))^{1-t_1} = \sum_{i=1}^{N} t_i \log y(w, x_i) + (1 - t_i) \log(1 - y(w, x_i)) \tag{18}$$

This can be solved by many different optimizations strategies. By optimizing this for each class we will find the optimal weights for the softmax-class separator in equation 15.

In the speaker recognition case we have tried using a soft-max function to separate the classes as described above by using it as a $P(C|x)$ approach. We have also used a full probabilistic model as described in using the $P(x|C)$ approach, by using a Gaussian mixture model this is described in the section: IV.

Intermediate result

III. Artificial Neural Network Classifier

e.g. graphical network model, training method, model flexibility (expressive power)

Introtext

Math

How we use it or why we don't use it

Intermediate result

IV. EM Classifier

e.g. training method, cost functions, model order selection, initialisation of parameters.

Introtext

Math

How we use it or why we don't use it

Intermediate result

V. Sequential Models

Markov model and Hidden Markov Model.
e.g. meaning of parameters, left-to-right model, outline of training/testing method.

Introtext

Math

How we use it or why we don't use it

Intermediate result

VI. Support Vector Machines

e.g. decision function, support vectors, soft margins, kernel trick.

Introtext

Math

How we use it or why we don't use it

Intermediate result

VII. Results

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

VIII. 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.

IX. Conclusion

References

[1] Md. Sahidullah and Goutam Saha. Design, analysis and experimental evaluation of block based transformation in MFCC computation for speaker recognition. *Speech Communication*, 54(4):543–565, may 2012.