Project: Automatic Speaker Recognition

Thomas Leichtle

August 15, 2013

1 Introduction

In the degree program for the M.Sc. in Computer Science, a course related project has to be done. I chose this one, because it's pratical and my interests in this field go back to the time before I started studying. Also even if the problem of automatic speaker recognition (ASR) is well known and solved in a sufficient way for most applications, it requires different steps that can be transfered to other classification problems. These steps are described in the following chapters, starting with feature extraction in chapter 2 over creating a model for data reduction in chapter 3 to methods for classification in chapter 4. In all three chapter first some general information is given and afterwards some examples are named and described.
After the theoretical part there are experimental results, experience reports and the description of the implementation as the practical part, for which a ASR program was implemented in C++.

2 Methods For Feature Extraction

Classification problems are usually solved using features. Their choice is an important factor for the performance and quality of classification. But how to choose the best features? It depends on the input data, in this case audio samples of speakers reading text and on the things to classify. The features should be invariant in respect of different changes. For example for ASR the features should be invariant to the spoken text, to volume and so on. Since ASR has been solved, there exist features with that properties. Both methods mentioned here try to model the vocal tract including tongue, teeth, etc. of

humans - or at least try to extract information correlated to it - because its shape determines how a person sounds.
For the calculation of the features some general preprocessing steps have to be done. Since the input data is a bytestream and the goal is to have features for each timestep, it is necessary to apply windowing - i.e. splitting up the stream into data blocks (frames) of length N. This makes sense, because we humans can also recognize a speaker already after a short time interval and not just at the end of a conversation. The parameter for configuration is the length of the frame. On one hand, if it's too short, tones might get cut off what results in bad recognition performance. But on the other hand, if it's too long, the features are smoothed over multiple tones what makes the recognition text dependent. For the implementation in chapter 5 a frame length of 32ms is used. Since the test-inputs are recorded with 16kHz, there are $N = 512$ samples per frame. The frames are created with an overlap as seen in figure 1. Later the feed $M = 256$ will be used.
After the windowing a hamming filter (figure 2) is applied. Therefore each frame is multiplied with $w(n) = 0.54 - 0.46 \cdot \cos(\frac{2\pi n}{N-1})$, $n \in \{0, \ldots, N-1\}$, i.e. each sample is weighted depending on its temporal distance to the center of the frame to smooth edge effects.

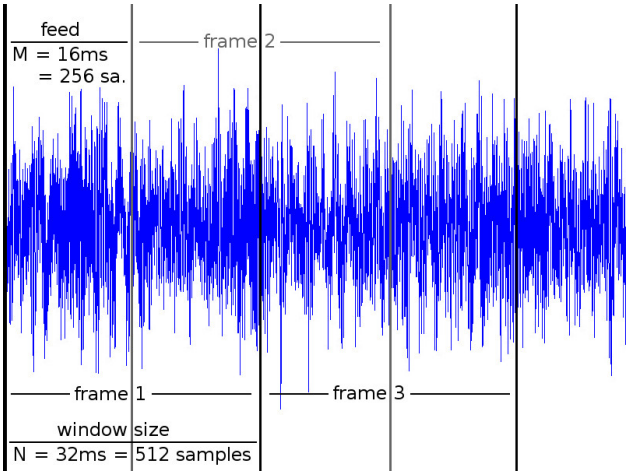

Figure 1: Windowing of input data. Create frames of length N with feed M.

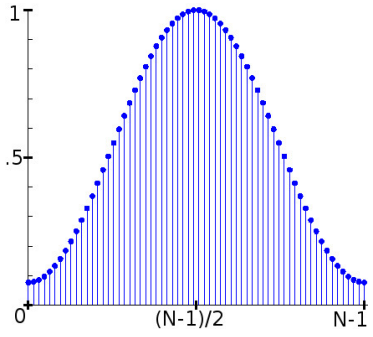

Figure 2:
Hamming window defined as $w(n) = 0.54 - 0.46 \cdot \cos(\frac{2\pi n}{N-1})$, $n \in \{0, \ldots, N-1\}$

2.1 Linear Prediction Coefficients

Some simple features for ASR are Linear Prediction Coefficients (LPCs) - see also [SS]. Given the datapoints of a frame $s(0), \ldots, s(N-1)$, take first p samples $s(0), \ldots, s(p-1)$ and represent $s(p)$ as a linear combination of them: $s(p) = a_p s(0) + a_{p-1} s(1) + \cdots + a_1 s(p-1)$. p is called the order. Shift one position and repeat this for the rest of the frame.

$$s(n) = \sum_{k=1}^{p} a_k s(n-k)$$

The resulting linear system of equations (LSE) has p variables and $N-p$ equations. Later $p = 12$ is used and therefore it is overdetermined. To minimize the error

$$E(a_1, \ldots, a_p) = \sum_n \left(s(n) - \sum_{k=1}^{p} a_k s(n-k) \right)^2$$

the partial derivations are set to zero.

$$\begin{aligned} \frac{\partial E}{\partial a_i} &= 0 \qquad i \in \{1, \ldots, p\} \\ \frac{\partial E}{\partial a_i} &= -2 \sum_n \left(s(n) - \sum_{k=1}^{p} a_k s(n-k) \right) s(n-i) = 0 \end{aligned}$$

This can be rewritten as

$$\sum_n s(n)s(n-i) = \sum_{k=1}^{p} a_k \sum_n s(n-k)s(n-i)$$

Using $r(i) := \sum_n s(n)s(n+i)$ leads to the LSE $r(i) = \sum_{k=1}^{p} a_k r(|k-i|)$

$$\begin{pmatrix} r(0) & r(1) & r(2) & \ldots & r(p-1) \\ r(1) & r(0) & r(1) & \ldots & r(p-2) \\ r(2) & r(1) & r(0) & \ldots & r(p-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r(p-1) & r(p-2) & r(p-3) & \ldots & r(0) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} r(1) \\ r(2) \\ \vdots \\ r(p) \end{pmatrix}$$

The matrix is symmetical with identical elements on the diagonal. Such matrices are called Toeplitz-Matrices and there are very efficient algorithms to solve such LSE, like the Durbin-Levinson algorithm. After calculation, the values $a_1, \ldots, a_p$ are the extracted features for this frame.

2.2 Mel Frequency Cepstral Coefficients

More advanced features in ASR are the Mel Frequency Cepstral Coefficients (MFCCs). They are more correlated to human hearing [Sch]. The procedure of extraction the mfccs can be roughly described by the following steps [Cam97]:

1. window the signal
2. take the Fast Fourier Transform (FFT)
3. take the magnitude
4. take the log
5. warp the frequencies according to the mel scale
6. take the inverse FFT

As mentioned in [Log00] the amplitude in step 3 can be taken since it is more important than the phase of the signal. The logarithm is used, because the perceived loudness is approximately logarithmic. In step 5, the different spectral components are binned and it is done according to the mel scale, because lower frequencies are more important for speech than higher. The inverse FFT at the end gives the mfcc features for the frame.
It won't be explained in more detail since existing libraries are used later on. For more information take a look at the references mentioned above.

3 Methods For Speaker Modelling

The next step in the process of ASR is the modelling of speakers. By processing a file of about 1 minute length with a frame size of 32ms and a feed of 16ms (see chapter 2) there are about 3750 feature vectors and it would be even more if more voice data for learning is available. That is the reason why smaller, more compact models need to be created.

3.1 Cluster Analysis & Competitve Learning

For the whole set of feature vectors S_i for each speaker i, a set of new feature vectors C_i with k prototypes, i.e. $|C_i| = k$ is needed. This is achieved by

using an incremental k-means algorithm for cluster analysis. In pseudocode it works as follows:

1. Define number of prototypes k and maxepoch
2. Initialize the k prototypes $c_1, \ldots, c_k$
   (e.g. choose random from given feature vectors)
3. While (epoch $<$ maxepoch) do
   - increment epoch
   - choose $x \in S_i$
   - calcuate $j = \text{argmin}_i(\|x - c_i\|)$ (winner detection)
   - $c_j = c_j + l(x - c_j)$ (winner update)

Instead of counting epochs it would also be possible to stop when the change $\|\Delta C\|$ after presenting N datapoints is smaller than a chosen ε. l is the learning rate and chosen as $l = 0.05$.

Applying the process for each speaker results in a set of codebooks C_i with k vectors each. Additionally to efficiency it creates some kind of fairness for voting, since now each speaker has the same amount of data, even if an uneven amount of input data was available for learning.

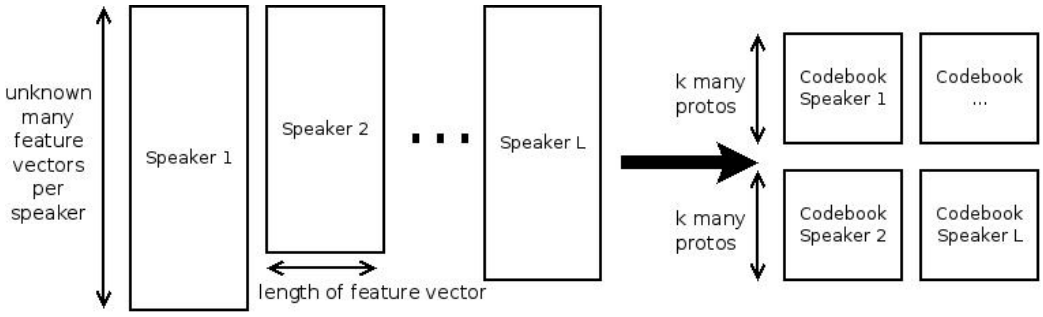

Figure 3: From input feature vector sets to codebooks.

4 Methods For Classification

Extracting the same features from new unknown inputs as for learning gives a set of feature vectors Y per input. An input can be a file or for live recognition also a sequence from a microphone.

4.1 Majority Voting

For each feature vector $y \in Y$ the most likely class is calculated. This is done by looking for the speaker i with the prototype c_j with the smallest euclidean distance to y and then this speaker gets a vote. After all vectors are classified the speaker with the most votes wins. It would be possible to evaluate the confidence of the decision - either by the ratio of received votes to all votes or by the distance to the speaker with the second most votes. Since the goal here is a single decision, the confidence is dropped.

5 Description Of Implementation

As the practical part of this project, implementation and evaluation of a software for ASR was done. There are two different variations. The first one is completely written in C/C++ and uses OpenMP (http://openmp.org/) for parallelization. LPC features are calculated with the Durbin-Levinson algorithm and is, as windowing, k-means and majority voting, self implemented. For MFCC features an approach using FFTW (http://fftw.org/) and libmfcc (http://code.google.com/p/libmfcc/) was developed, but due to performance issues a second version of the program was created.
The second variant uses openSMILE [EWS10] for all preprocessing steps till mfccs. Compression with k-means and classification is done the same way as in the first variant.

6 Statistical Experiments

For evaluation a dataset of 10 speakers with 3 files of length > 60 seconds was chosen. They were recorded with 1 channel at 16kHz sampling rate. 2 of those files were used for learning. From the last file random snippets of different length (1sec, 2sec, 5sec, 10sec) were cut out and classified. Additional parameters were window/frame size N = 32ms = 512samples, feed M = 16ms = 256samples and maxepoch = 100 for kmeans. The results are shown in the following subsections.

6.1 LPC features

The order was $p = 12$ for LPC calculation. Each of the tables in figure 4 was created by randomly taking 1000 snippets of given length from the classification set.

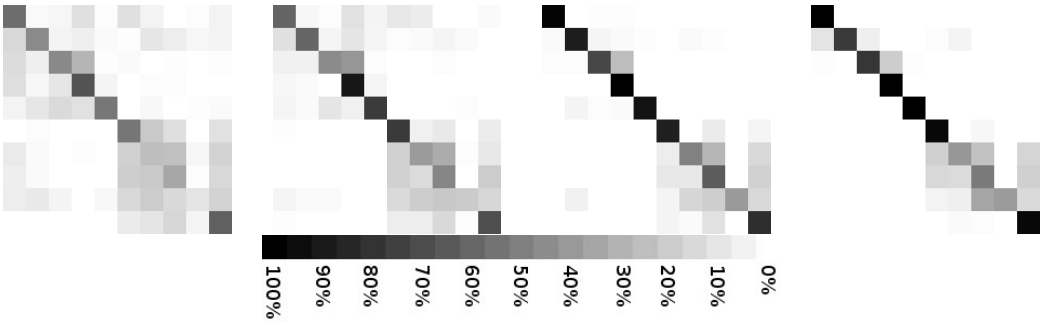

Figure 4: LPC features: Each row and column represents one speaker. It shows how likely it is that speaker i (vertical) is classified as speaker j (horizontal). $k = 50$ prototypes are used in compression and the length of the input data for classification varies from 1 second on the left, over 2 sec, 5 sec to 10 seconds on the right.

In figure 4 it is good to see how the rate of correct classification increases with the number of frames for decision. 1 second includes about 60 frames which are classified and used for the majority voting, where 10 seconds give about 600 votes. But even then there is a relatively high error rate for speaker 7 to 9, indicated by the gray pixels not on the diagonal.

6.2 MFCC features

For MFCC features 13 frequency bins have been used. Again each of the tables in figure 4 was created by randomly taking 1000 snippets of given length from the classification set.

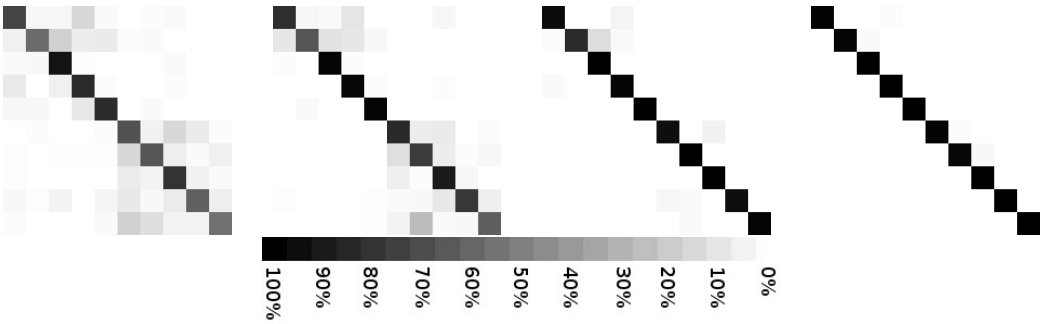

Figure 5: MFCC features: Each row and column represents one speaker. It shows how likely it is that speaker i (vertical) is classified as speaker j (horizontal). $k = 50$ prototypes are used in compression and the length of the input data for classification varies from 1 second on the left, over 2 sec, 5 sec to 10 seconds on the right.

In figure 5, like in figure 4, the correct classification rate increases with the length of the snippets. But this time it reaches nearly 100 percent accuracy. This shows the advantage of mfcc over lpc for text independent automatic speaker recognition. But as mentioned before it is easier and faster to calculate LPCs.

The classification rate also varies with other features of the creation of the model, like the number of prototypes, learning rate and maxepoch. Since this is only a proof of concept to get a deeper understanding, no further optimization of these parameters is done.

7 Experience Live-Recognition

The live system is based on the second version of the program, the one with openSMILE. It supports reading from microphone by portaudio (http://portaudio.com) and therefore works the same way as the offline version. The speaker models are created by prerecorded offline data and only new data for classification is recorded on the fly. For the majority voting a predefined number of frames per decision is taken, e.g. 75 frames per decision. Less frames result in a faster and less accurate decision, while more lead to a more accurate one. The problem using more frames is that the accuracy is lost if the speaker stops talking within the frame.
For pratical reasons a sequence of 'silence' was recorded as an additional speaker. This prevents the system to make unconfident decision for any speaker if noone is talking and results in the choice of this silent speaker.
The program works very well if only low background noise is present. With increasing noise the correct classification rate decreases as expected.

References

[Cam97] J. P. Campbell. Speaker recognition: A tutorial, 1997.

[EWS10] Florian Eyben, Martin Wöllmer, and Björn Schuller. Opensmile: the munich versatile and fast open-source audio feature extractor. In *Proceedings of the international conference on Multimedia*, MM '10, pages 1459–1462, New York, NY, USA, 2010. ACM.

[Log00] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *International Symposium on Music Information Retrieval*, 2000.

[Sch] Stefan Scherer. Mfcc implementierung. http://www.informatik.uni-ulm.de/ni/Lehre/SS06/PraktikumNI/MFCC.pdf.

[SS] Alfred Strey and Friedhelm Schwenker. Folien zum lpc. http://www.informatik.uni-ulm.de/ni/Lehre/SS06/PraktikumNI/LPC.ps.