Speaker Recognition:

Special Course
IMM•DTU

Lasse L Mølgaard
s001514

Kasper W Jørgensen
s001498

December 14, 2005

Contents

1 Introduction

Speaker recognition has been an interesting research field for the last decades, which still yields a number of unsolved problems.

Speaker recognition is basically divided into speaker identification and speaker verification. Verification is the task of automatically determining if a person really is the person he or she claims to be. This technology can be used as a biometric feature for verifying the identity of a person in applications like banking by telephone and voice mail. The focus of this project is speaker identification, which consists of mapping a speech signal from an unknown speaker to a database of known speakers, i.e. the system has been trained with a number of speakers which the system can recognize. The systems can be subdivided into text-dependent and text-independent methods. Text-dependent systems require the speaker to utter a specific phrase (pin-code, password etc.), while a text-independent method should catch the characteristics of the speech irrespective of the text spoken.

Speaker identification has been done successfully using Vector Quantization (VQ). This technique consists of extracting a small number of representative feature vectors as an efficient means of characterizing the speaker-specific features. Using training data these features are clustered to form a speaker-specific codebook. In the recognition stage, the test data is compared to the codebook of each reference speaker and a measure of the difference is used to make the recognition decision. The process is depicted in figure 1.

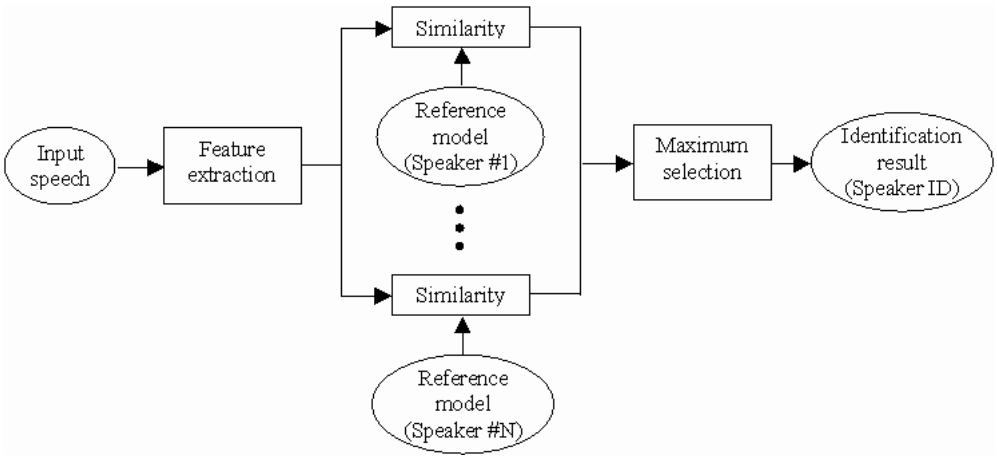


Figure 1: Conceptual presentation of speaker identification. Figure from [3]

The VQ in this project is done utilizing Mel Frequency Cepstral Coefficients and Linear Prediction Cepstral Coefficients and a simple clustering scheme using the k-means algorithm, based on the ideas presented in [3] and [7].

2 Speech Feature Extraction

Feature extraction in a classification problem is about reducing the dimensionality of the input-vector while maintaining the discriminating power of the signal. We know from 'the curse of the dimensionality' that the number of training/test-vectors needed for a classification problem grows exponential with the dimension of the given input-vector, so clearly feature extraction is needed.

When dealing with speech signals there are some criteria that the extracted features should meet. Some of them are listed below [6]:

- discriminate between speakers while being tolerant of intra-speaker variabilities,
- easy to measure,
- stable over time,
- occur naturally and frequently in speech,
- change little from one speaking environment to another,
- not be susceptible to mimicry.

For speech signals it is known that the best features is based on spectral analysis. The reason for that is, that the speech signal can be estimated with a linear superposition of sine-waves with different amplitudes and phases. In our project we have been using Linear Prediction Cepstral Coefficients and Mel Frequency Cepstral Coefficients as features for the classification problem. These methods are described below.

2.1 Framing and Windowing

The speech signal is slowly varying over time (quasi-stationary), that is when the signal is examined over a short period of time (5-100msec), the signal is fairly stationary. Therefore speech signals are often analyzed in short time segments, which is referred to as *short-time spectral analysis*.

This practically means that the signal is blocked in frames of typically 20-30 msec. Adjacent frames typically overlap each other with 30-50%, this is done in order not to lose any information due to the windowing.

After the signal has been framed, each frame is multiplied with a window function $w(n)$ with length N, where N is the length of the frame. Typically the Hamming window is used:

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right), 0 \leq n \leq N-1$$

The windowing is done to avoid problems due to truncation of the signal.

2.2 Cepstrum

As described in [2] the speech signal is composed of a quickly varying part $e(n)$(excitation sequence) convolved with a slowly varying part $\theta(n)$ (vocal system impulse response):

$$s(n) = e(n) * \theta(n)$$

The convolution makes it difficult to separate the two parts, therefore the cepstrum is introduced. The cepstrum is defined in the following way:

$$c_s(n) = \mathfrak{F}^{-1}\Big\{log|\mathfrak{F}\{s(n)\}|\Big\}$$

$\mathfrak{F}$ is the DTFT and $\mathfrak{F}^{-1}$ is the IDTFT. By moving the signal to the frequency-domain, the convolution becomes a multiplication:

$$S(\omega) = E(\omega)\Theta(\omega)$$

Further, by taking the logarithm of the spectral magnitude the multiplication becomes an addition:

$$log|S(\omega)| = log|E(\omega)\Theta(\omega)| = log|E(\omega)| + log|\Theta(\omega)| = C_e(\omega) + C_\theta(\omega)$$

The Inverse Fourier Transform is linear and therefore work individually on the two components:

$$c_s(n) = \mathfrak{F}^{-1}\big\{C_e(\omega) + C_\theta(\omega)\big\} = \mathfrak{F}^{-1}\big\{C_e(\omega)\big\} + \mathfrak{F}^{-1}\big\{C_\theta(\omega)\big\} = c_e(n) + c_\theta(n)$$

The domain of the signal $c_s(n)$ is called the *quefrency*-domain. Figure 2 shows the speech signal transformation process.

2.3 Linear Prediction Cepstral Coefficients

One way to extract features is to use the *Linear Prediction Analysis* and convert it to Cepstral Coefficients (called LPCC). The idea behind this method is that a given speech sample can be approximated with a linear combination of the past p speech samples[5]:

$$\hat{s}_n = -\sum_{k=1}^{p} a_k s_{n-k}$$

The coefficients a_k are called the LP coefficients and are found using the *Levinson-Durbin recursion*[2]. p is the so called prediction order. The p LP-coefficients are then converted to Q cepstral coefficients using the following equations:

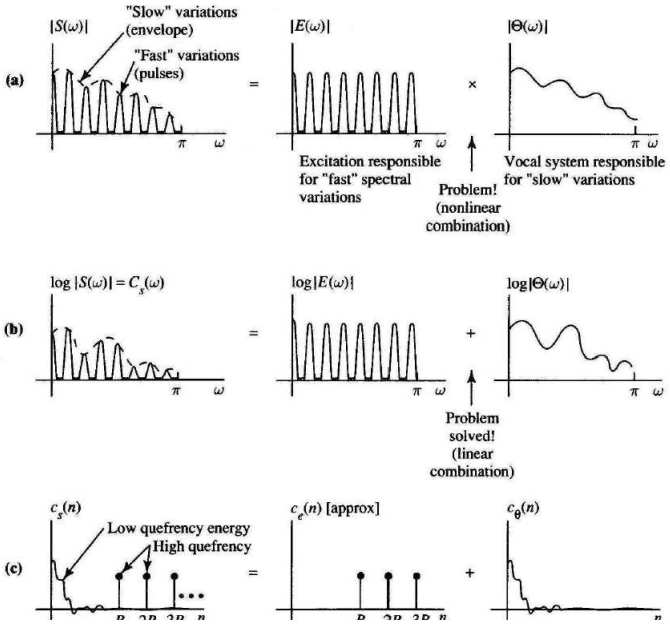


Figure 2: Motivation for using cepstrum. Figure taken from [2]

$$c_1 = a_1 \tag{1}$$

$$c_n = \sum_{k=1}^{n-1}(1-k/n)a_k c_{n-k} + a_n, \qquad 1 < n \leq p \tag{2}$$

$$c_n = \sum_{k=1}^{n-1}(1-k/n)a_k c_{n-k}, \qquad n > p \tag{3}$$

The cepstral sequence is weighted by a window function $\omega_c(i)$ of the form:

$$\omega(i) = 1 + \frac{Q}{2} sin\Big(\frac{\pi i}{Q}\Big), \qquad i = 1, 2, ..., Q \tag{4}$$

2.4 Mel-frequency Cepstral Coefficients

The cepstral coefficients described above have been used with success in speech recognition applications. A further improvement to this method can be obtained by using the 'mel-based cepstrum' or mel-cepstrum for short. The mel-cepstrum is calculated in the same way as the real cepstrum except that the frequency scale is warped to correspond to the mel scale.

The mel scale is based on an empirical study of the human perceived pitch or frequency. The scale is divided into the units "mel"s. The test persons in the study started out hearing a frequency of 1000 Hz, which was labeled 1000 mels for reference. The persons were then asked to change the frequency until they perceived the frequency to be twice the reference. This frequency was then labeled 2000 mels. The test was then repeated with half the frequency, $\frac{1}{10}$, 10 and so on, labeling these frequencies 500 mels, 100 mels, and 10000 mels. Based on these results a mapping of the normal frequency scale to the mel scale was possible.

The mel scale is generally speaking a linear mapping below 1000 Hz and logarithmically spaced above. The mapping is usually done using an approximation (where f_{mel} is the perceived frequency in mels), taken from [4]:

$$f_{mel} = 2595 * log_{10}(1 + \frac{f}{700})$$

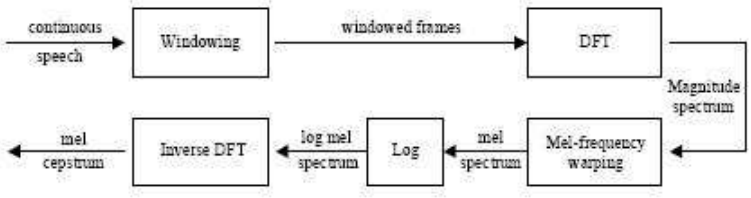


Figure 3: MFCC calculation

The calculation of the mel cepstral coefficients is illustrated in figure 3. The mel frequency warping is most conveniently done by utilizing a filter bank with filters centered according to mel frequencies, as seen in figure 4. The width of the triangular filters vary according to the mel scale, so that the log total energy in a critical band around the center frequency is included.

All in all the result after warping is a number of coefficients Y(k):

$$Y(k) = \sum_{j=1}^{N/2} S(j)H_k(j) \tag{5}$$

The last step of the cepstral coefficient calculation is to transform the log of the quefrency domain coefficients to the frequency domain. For this we utilize the IDFT, where N' is the length of the DFT used previously:

$$c(n) = \frac{1}{N'} \sum_{k=0}^{N'-1} Y(k) e^{j\frac{k2\pi}{N'}n} \tag{6}$$

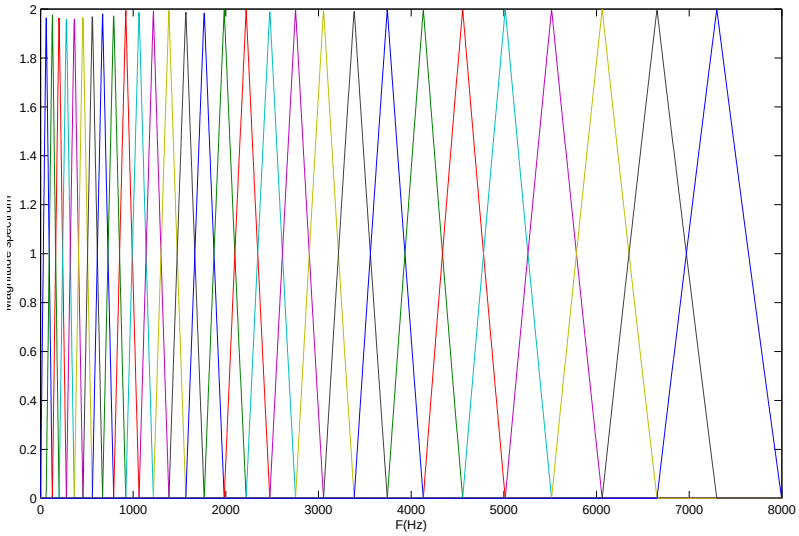


Figure 4: Mel spaced filter bank w. 29 filters

Which can be simplified, because Y(k) is real and symmetric about $N'/2$, by replacing the exponential by a cosine:

$$c(n) = \frac{1}{N'} \sum_{k=0}^{N'-1} Y(k) \cos(\frac{k2\pi}{N'} n) \tag{7}$$

2.5 Delta Cepstrum

To catch the changes between the different frames the differenced or delta cepstrum is used. It is simply defined as:

$$\Delta c_s(n;m) = \frac{1}{2}(c_s(n;m+1) - c_s(n;m-1)), \qquad i = 1, 2, ..., Q \tag{8}$$

3 Vector Quantization

Speaker recognition is the task of comparing an unknown speaker with a set of known speakers in a database and find the best matching speaker.

3.1 Speaker Database

The first step is to build a speaker-database $C_{database} = \{C_1, C_2, ..., C_N\}$ consisting of N codebooks, one for each speaker in the database. This is done by first converting the raw input signal into a sequence of feature vectors $X =$

$\{\mathbf{x}_1, ..., \mathbf{x}_T\}$. These feature vectors are clustered into a set of M codewords $C = \{\mathbf{c}_1, ..., \mathbf{c}_M\}$. The set of codewords is called a codebook. The clustering is done by a clustering-algorithm, in this project we are using the *K-means algorithm* which is described below.

3.2 K-means

The *K-means algorithm* partitions the T feature vectors into M centroids. The algorithm first chooses M cluster-centroids among the T feature vectors. Then each feature vector is assigned to the nearest centroid, and the new centroids are calculated. This procedure is continued until a stopping criterion is met, that is the mean square error between the feature vectors and the cluster-centroids is below a certain threshold or there is no more change in the cluster-center assignment.

3.3 Speaker Matching

In the recognition phase an unknown speaker, represented by a sequence of feature vectors $\{\mathbf{x}_1, ..., \mathbf{x}_T\}$, is compared with the codebooks in the database. For each codebook a distortion measure is computed, and the speaker with the lowest distortion is chosen,

$$C_{best} = \underset{1 \leq i \leq N}{argmin} \{s(X, C_i)\}$$

One way to define the distortion measure is to use the average of the Euclidean distances:

$$s(X, C_i) = \frac{1}{T} \sum_{t=1}^{T} d(\boldsymbol{x}_t, \boldsymbol{c}_{min}^{i,t}),$$

where $\boldsymbol{c}_{min}^{i,t}$ denotes the nearest codeword $\boldsymbol{x}_t$ in the codebook C_i and $d(.)$ is the Euclidean distance. Thus, each feature vector in the sequence X is compared with all the codebooks, and the codebook with the minimized average distance is chosen to be the best.

3.4 Weighting Method

Franti and Kinnunen [7] propose a weighting method that takes the correlation between the known speakers in the database into account. The idea is that larger weights should be assigned to vectors that has higher discriminating power. If vectors from more codebooks are very close in feature space it is not so obvious which one of the vectors that a given unknown vector belongs to. On the other hand if a vector is far from the other vectors of the other codebooks, then it is more clear which codebook the given unknown vector belongs to.

Thus, the following algorithm are proposed to assign weights to all codewords in the database:

```
PROCEDURE ComputeWeights(S: SET OF CODEBOOKS) RETURN WEIGHTS
FOR EACH C_i IN S DO         % Loop over all codebooks
    FOR EACH c_j IN C_i DO   % Loop over all codebooks
        sum := 0
        FOR EACH C_k, k!=i, IN S DO                % Find nearest code vector_
            d_min := DistanceToNearest(c_j, C_k);  % _ from all other codebooks
            sum := sum + 1/d_min;
        ENDFOR;
        w(c_ij) := 1/sum;
    ENDFOR;
ENDFOR;
```

Instead of using a distortion measure, a similarity measure that should be maximized are considered:

$$s_w(X, C_i) = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{d(\boldsymbol{x}_t, \boldsymbol{c}_{min}^{i,t})}w(\boldsymbol{c}_{min}^{i,t})$$

The experimental results from [7] shows better recognition rate when using weights.

4 Data

The methods presented above have been tested using the ELSDSR (English Language Speech Database for Speaker Recognition), which is thoroughly described in [4]. The database consists of 22 speakers, whereof 10 are female and 12 are male, and the ages span from 24 to 63 years. 20 of the speakers are Danish natives, 1 Icelandic and 1 Canadian.

The data is divided into two parts, i.e. a training part, with sentences made to attempt to capture all the possible pronunciation of English language, which includes the vowels, consonants and diphthongs, and a test set of random sentences. The training set consists of seven paragraphs, which include 11 sentences; and forty-four sentences for test. Shortly there are 154 (7*22) utterances in the training set; and for the test set, 44 (2*22) utterances are provided. On average, the duration for reading the training data is: 78.6s for male; 88.3s for female; 83s for all. And the duration for reading test data, on average, is: 16.1s (male); 19.6s (female); 17.6s (for all). The duration of the training shots vary from 66.2s to 102.9s; and from 9.3s to 25.1.

The training of the models was done using all seven paragraphs for each speaker, while each test utilized one paragraph from the test set providing 44 tests.

5 Results

The different methods described have been quite extensively explored using the data described above. The aspects evaluated are:

- Sweep of parameters in feature extraction
- Evaluation of MFCC and LPCC on different test shot lengths
- Addition of delta cepstrum coefficients
- Effect of additive noise on test set

The tests have been done according to the description above. The tests were done using the functions implemented in the voicebox matlab package [1].

5.1 Parameters of the MFCC

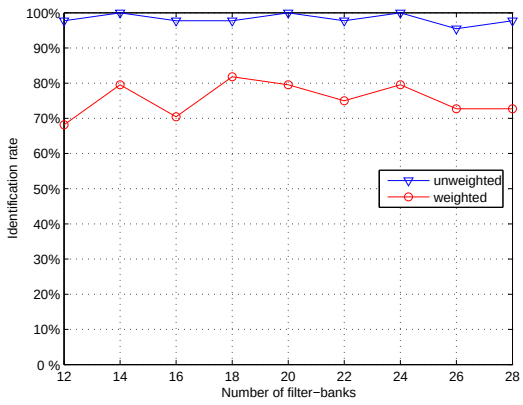


Figure 5: Performance evaluation with 12 MFCC as function of number of filter-banks with 12 MFCC. The codebook size is 8.

Calculation of the MFCC's has a number parameters that can be varied. The first aspect to be investigated is how many filters to use in the filter bank. To keep the calculation times manageable we have chosen to use 12 coefficients. With this constraint, the main parameter to change is the number of filters in the filter bank. Figure 5 shows the performance using a codebook size of 8.

The figure does not show anything conclusive about the how to choose the number of filters, one of the factors is namely that the training relies on the random procedure k-means which might produce differing results on different runs.

5.2 MFCC vs. LPCC

To evaluate the two features, the performance on different test shot lengths were conducted. Using all test persons, three tests were made;

- Using the full test shots
- A 2s shot, starting at t=2s
- A 0.2s shot, starting at t=2s

The shorter shots start after 2s to avoid silent periods in the beginning of the recordings. Of course, the shots might not contain any speech data anyway, but this has not been investigated further. The LPCC calculation showed some numerical problems, when the signals contained long segments of zeros. To counteract this, some gaussian noise with a standard deviation of 0.0001 was added.

The test uses 12 MFCC with 29 filters, and 12 LPCC's using 12th order LP-analysis. The test run varies the size of the codebook (i.e. the number of codewords assigned to each speaker). The codebook size increments are powers of 2 to reproduce the results presented in [7].

The figures 6, 7 and 8 show that the purely euclidian distance measure clearly outperforms the weighting scheme in all cases. Using the whole test shot shows that both MFCC and LPCC perform perfect identification using 16 and 4 codewords for each speaker, respectively. The 2s test shot shows almost the same performance. The short 0.2s test shot shows that the MFCC features give a 73% identification rate while the LPCCs only shows 60%.

5.3 Delta coefficients

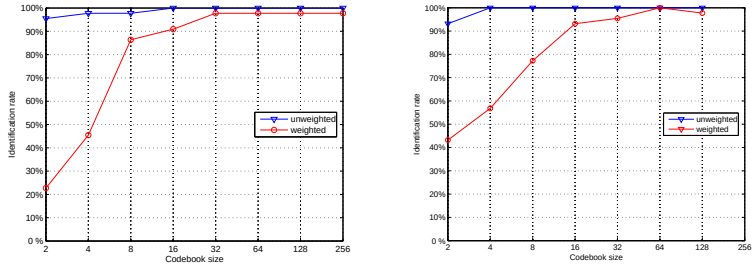
The above test was repeated with the addition of the delta coefficients presented in section 2.5. The test runs were limited to with a codebook size of 2 to 128 to save computation time.

The results seen in figures 9, 10 and 11 show that perfect identification is achieved with full test shots, even though at a larger codebook size than above, at least for MFCCs. The same tendency is apparent at shorter test shots, but still the results are comparable to those achieved without delta coefficients.

5.4 Noise standard deviation

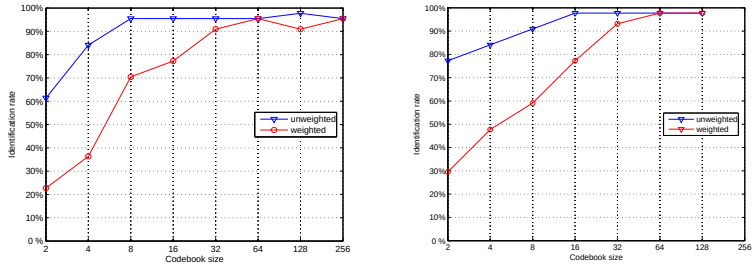
An important property of the features is the ability to cope with noise. In general we can apply a white noise signal to the speech signal and still recognise the speaker anyway. *To test the* ability *of features against noise* The test setup was:

- 12 MFCCs using 29 filters and 12 LPCCs using 12th order LP analysis



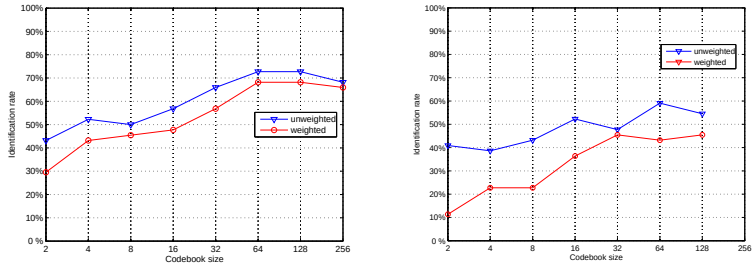
(a) (b)

Figure 6: Performance for varying codebook sizes for full test shots using (a) 12 MFCCs and (b) 12 LPCCs



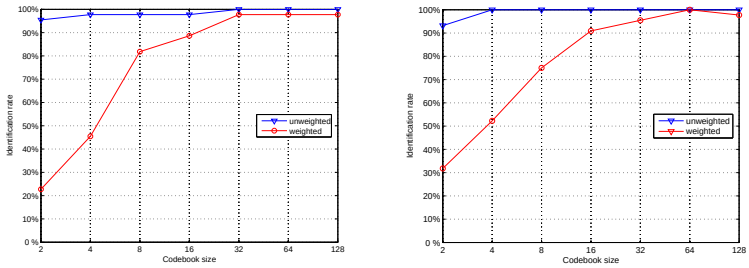
(a) (b)

Figure 7: Performance for varying codebook sizes for 2s test shots using (a) 12 MFCCs and (b) 12 LPCCs



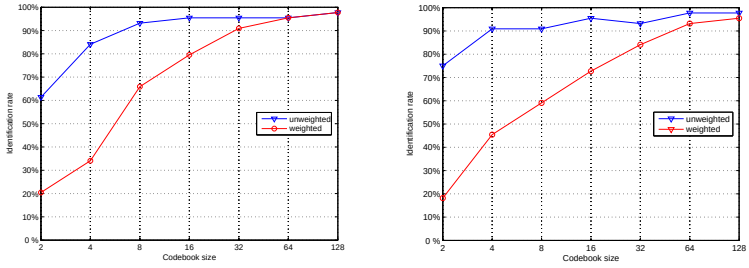
(a) (b)

Figure 8: Performance for varying codebook sizes for 0.2s test shots using (a) 12 MFCCs and (b) 12 LPCCs



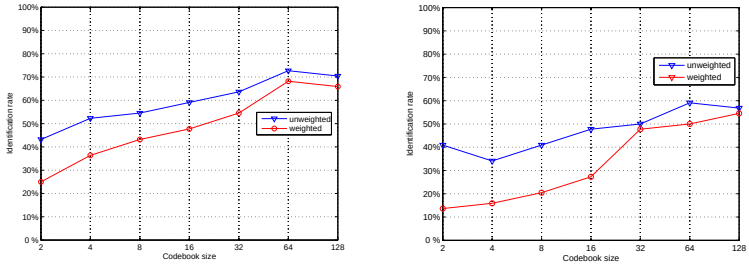
(a) (b)

Figure 9: Performance for varying codebook sizes for full test shots using (a) 12 MFCCs and 12 delta coefficients (b) 12 LPCCs and 12 delta coefficients



(a) (b)

Figure 10: Performance for varying codebook sizes for 2s test shots using (a) 12 MFCCs and 12 delta coefficients (b) 12 LPCCs and 12 delta coefficients

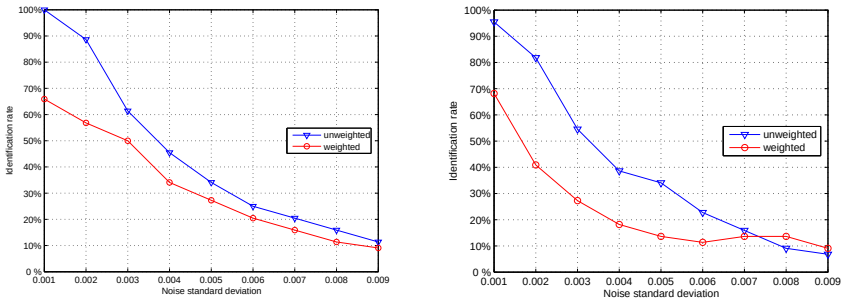


(a) (b)

Figure 11: Performance for varying codebook sizes for 0.2s test shots using (a) 12 MFCCs and 12 delta coefficients (b) 12 LPCCs and 12 delta coefficients

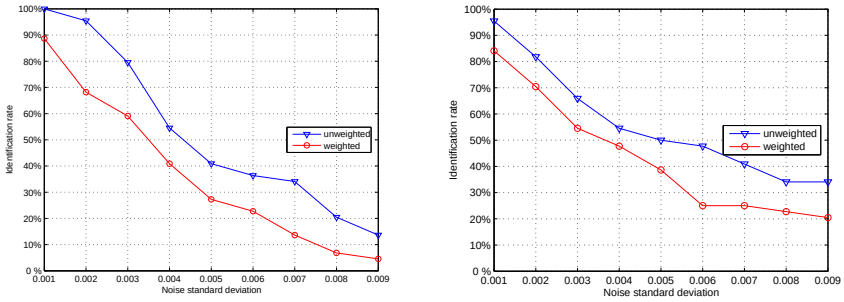
- Codebook size of 8 and 16
- Additive gaussian noise $X \sim (0, \sigma^2)$ with $\sigma \in [0.001; 0.009]$

The figure 12 and 13 show that the noise clearly influences on the performance of the system, making the classification almost useless at high noise levels. It seems that the LPCCs are most resistant to low noise levels, while the MFCCs have a little better performance at larger noise levels, using a codebook size of 16. Increasing the code book size from 8 to 16 shows a definite improvement, especially at the higher noise levels.



(a) LPCC (b) MFCC

Figure 12: Performance of coefficients to noise. The tests use (a) 12 LPCCs and (b) 12 MFCCs and a codebook size of 8. The noise std dev. is changed over the range [0.001;0.009]



(a) LPCC (b) MFCC

Figure 13: Performance of coefficients to noise. The tests use (a) 12 LPCCs and (b) 12 MFCCs and a codebook size of 16. The noise std dev. is changed over the range [0.001;0.009]

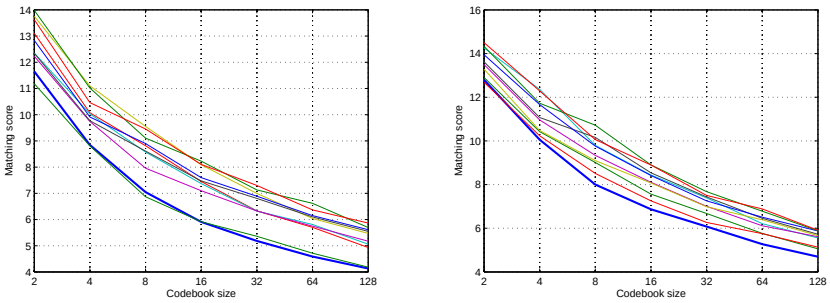
5.5 Decision Certainty

To see how confident our decisions are we have made some plots of the distortion measure as a function of codebook size. In the plots the correct speaker's distortion measure is marked with a thick line, the other lines represent the distortion measures for the 9 speakers with the lowest measure. The plots are made with both 12 Mel Frequency Cepstral Coefficients calculated using 29 filter-banks and 12 Linear Prediction Cepstral Coefficients calculated using 12th order LP analysis.

In figure 14 the distortion measures for the test speech sample `FEAB_Sr5.wav` are shown. This particular speech sample has shown, during our different tests, to be the most difficult to recognize correct. The MFCC distortion measure from the correct speaker are very close to another speaker (`FAML`). The distortion measures from these two speakers are well separated from the other speakers. When using LPCC the right speaker are better separated from the runner up.

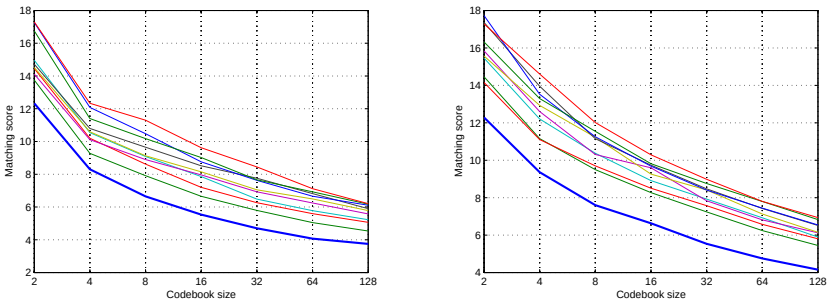
In figure 15 a randomly chosen test speaker (`FUAN_Sr39.wav`) are shown for reference. This figure also show a slightly better separation when using LPCC.

Another thing to see from these figures are that the difference in distortion measure between the correct speaker and the runner up is almost the same when varying the codebook size.



(a) MFCC (b) LPCC

Figure 14: Distortion measure for test speech sample `FEAB_Sr5.wav` as function of codebook size. Thick line is distortion for correct speaker, rest are the 9 speakers with lowest distortion.



(a) MFCC (b) LPCC

Figure 15: Distortion measure for test speech sample `FUAN_Sr39.wav` as function of codebook size. Thick line is distortion for correct speaker, rest are the 9 speakers with lowest distortion.

6 Conclusion

The goal of this project was to implement a text-independent speaker recognition system. Further, the aim was to investigate different feature extraction methods and their impact on the recognition rate.

The feature extraction is done using Mel Frequency Cepstral Coefficients (MFCC) and Linear Prediction Cepstral Coefficients (LPCC). The speakers was modeled using Vector Quantization (VQ). Using the extracted features a codebook from each speaker was build by clustering the feature vectors. The clustering was done using the K-means algorithm. Codebooks from all the speakers was collected in a speaker database. Two different distortion measures was used when matching an unknown speaker with the speaker database. The first method is based on minimizing the Euclidean distance. The second method was suggested by [7]. This method is based on maximizing the inverse Euclidean distance combined with a weight measure.

The experiments conducted showed that it was possible to obtain 100% identification rates for both MFCC and LPCC based features. The perfect identification was done using the full training set of the ELSDSR ddatabase and full test shots. Reducing the test shot lengths reduced the recognition rate giving a maximal rate of 97% for 2s shots and 73% for 0.2s shots, with MFCCs giving slightly better results. Adding delta coefficients to the feature set did not show any improvements. The systems were tested in a setting with noise added to the test signals, demonstrating the susceptibility to noise. This showed a slight advantage for the LPCCs. A inspection of the distortion measures showed that the difference between the correct speaker and the runner up did not vary with higher codebook size.

The two different distortion measures were used in the tests, which

showed that the purely Euclidean measure outperformed the weighting scheme in all cases.

All in all the project has shown that VQ using cepstral features is an simple and efficient way to do speaker identification. The results did not show any conclusive evidence of whether to use LPCC or MFCC features.

References

[1] M. Brooks, "Voicebox: Speech Processing Toolbox for MATLAB," `http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/`.

[2] J. R. Deller, J. G. Proakis and J. H. L. Hansen, **Discrete-time Processing of Speech Signals**, Prentice Hall, New Jersey, 1993.

[3] M. N. Do, "Digital Signal Processing Mini-Project: An Automatic Speaker Recognition System," `http://lcavwww.epfl.ch/~minhdo/asr_project/`.

[4] L. Feng, **Speaker Recognition**, Master's thesis, Technical University of Denmark, Informatics and Mathematical Modelling, 2004, ISSN: 1601-233X.

[5] J. P. C. Jr, "Speaker Recognition: A Tutorial," in **Proceedings of the IEEE, vol. 85 no. 9**, 1997.

[6] E. Karpov, **Real-Time Speaker Identification**, Master's thesis, University of Joensuu Department of Computer Science, 2003.

[7] T. Kinnunen and P. Franti, "Speaker Discriminative Weighting Method for VQ-based Speaker identification," 2001.

A Matlab code

A.1 testnoise_cc.m

```
clear;

voiceboxpath='~/pep/voicebox';
addpath(voiceboxpath);

[train.data test.data] = load_data;
train.mfcc        = cell(size(test.data,1),1);
train.kmeans.x    = cell(size(test.data,1),1);
train.kmeans.esql = cell(size(test.data,1),1);
train.kmeans.j    = cell(size(test.data,1),1);

fs         = 16000;
C          = 16;       % number of cluster centers in K-means
persons    = 22;

disp('Calculating CCs for training set...')
for i=1:size(train.data,1)
    i
    temp = [];
    for s=1:size(train.data,2)
        temp = [temp;train.data{i,s}];
    end
    noise = rand(length(temp),1)*0.0001;
    cepstral = cc(temp+noise,256,128,12,12);  % find the cepstral
        coefficients
    train.mfcc{i} = cepstral';
end

disp('Performing K-means...')
for i=1:size(train.data,1)
    i
    [train.kmeans.j{i} train.kmeans.x{i}] = kmeans(train.mfcc{i}(:,1:12),C);
end

disp('compute weights')
w = computeweights(train.kmeans.x);

weighted=zeros(9,1);
unweighted=zeros(9,1);

for ite = 1:9
    correct=0;
    correctweight =0;

    disp('Calculating CCs for test set...')
    for i=1:size(test.data,1)
        i
        for s=1:size(test.data,2)
            noise = randn(length(test.data{i,s}),1)*0.001*ite;
            cepstral = cc(test.data{i,s}+noise,256,128,12,12);  % find the
                cepstral coefficients
            test.mfcc{i,s} = cepstral';
        end
    end

    for i = 1:persons
        for s=1:2
```

```
            mins        = inf;
            minsweight = 0;

            for x=1:persons  % Run for all codebooks
                disteu = disteusq(train.kmeans.x{x}, test.mfcc{i,s}(:,1:12),
                    'x');
                sdist(i,s,x) = sum(min(disteu)) / size(disteu,2); % calc
                    distortion without weights

                [cmin cminindex] = min(disteu);
                sdistweight(i,s,x) = sum(w(x,cminindex)./cmin) / size(disteu
                    ,2); %calc distortion with weights

                % find best match wihtout weights
                if sdist(i,s,x) < mins
                    mins = sdist(i,s,x);
                    index = x;
                end
                % find best match with weights
                if sdistweight(i,s,x) > minsweight
                    minsweight = sdistweight(i,s,x);
                    indexweight = x;
                end
            end
            [i index]
            if i == index
                correct = correct+1;
            end
            if i == indexweight
                correctweight = correctweight + 1;
            end

            unweightedgem(i,s) = index;
            weightedgem(i,s)   = indexweight;

        end
    end
    unweighted(ite)=correct/(persons*2)
    weighted(ite)=correctweight/(persons*2)
end
```

A.2 testnoise_mfcc.m

```
clear;

voiceboxpath='..\pep\pep\voicebox\';

addpath(voiceboxpath);

[train.data test.data] = load_data;
train.mfcc         = cell(size(test.data,1),1);
train.kmeans.x     = cell(size(test.data,1),1);
train.kmeans.esql  = cell(size(test.data,1),1);
train.kmeans.j     = cell(size(test.data,1),1);

fs         = 16000;
C          = 8;        % codebook size
persons    = 22;

disp('Calculating MFCCs for training set...')
for i=1:size(train.data,1)
    i
    temp = [];
    for s=1:size(train.data,2)
        temp = [temp;train.data{i,s}];
    end
    mels = melcepst(temp,fs,'x');  % find the cepstral coefficients
    train.mfcc{i} = mels;
end

disp('Performing K-means...')
for i=1:size(train.data,1)
    i
    [train.kmeans.j{i} train.kmeans.x{i}] = kmeans(train.mfcc{i},C);  % use
        matlab's own kmeans
end

disp('compute weights')
w = computeweights(train.kmeans.x);

weighted=zeros(9,1);
unweighted=zeros(9,1);

for ite = 1:9
    correct=0;
    correctweight =0;

    disp('Calculating MFCCs for test set...')
    for i=1:size(test.data,1)
        i
        for s=1:size(test.data,2)
            noise = randn(length(test.data{i,s}),1)*0.001*ite;    % add noise
                to signal
            mels = melcepst(test.data{i,s}+noise,fs,'x');
            test.mfcc{i,s} = mels;
        end
    end


    for i = 1:persons
        for s=1:2
            mins       = inf;
            minsweight = 0;
```

```
            for x=1:persons  % Run for all codebooks
                disteu = disteusq(train.kmeans.x{x}, test.mfcc{i,s},'x');
                sdist(ite,i,s,x)   = sum(min(disteu)) / size(disteu,2); %
                    calc distortion without weights

                [cmin cminindex] = min(disteu);
                sdistweight(ite,i,s,x) = sum(w(x,cminindex)./cmin) / size(
                    disteu,2); % calc distortion with weights

                %find best match wihtout weights
                if sdist(ite,i,s,x) < mins
                    mins = sdist(ite,i,s,x);
                    index = x;
                end
                %find best match with weights
                if sdistweight(ite,i,s,x) > minsweight
                    minsweight = sdistweight(ite,i,s,x);
                    indexweight = x;
                end
            end
            [i index]
            if i == index
                correct = correct+1;
            end
            if i == indexweight
                correctweight = correctweight + 1;
            end

            unweightedgem(i,s,ite) = index;
            weightedgem(i,s,ite)   = indexweight;
        end
    end
    unweighted(ite) =correct/(persons*2)
    weighted(ite)   =correctweight/(persons*2)
end
```

A.3 load_data.m

```
function [train,test] = load_data

traindir = '../pep/madam_skrald/elsdsr/train/';
testdir = '../pep/madam_skrald/elsdsr/test/';


initial = ['FAML';'FDHH';'FEAB';'FHRO';'FJAZ';'FMEL';'FMEV';...
           'FSLJ';'FTEJ';'FUAN';'MASM';'MCBR';'MFKC';'MKBP';...
           'MLKH';'MMLP';'MMNA';'MNHP';'MOEW';'MPRA';'MREM';'MTLS'];

sentence = ['a' 'b' 'c' 'd' 'e' 'f' 'g'];

filename = cell(44,1);
filename = {'FAML_Sr3.wav' 'FAML_Sr4.wav' 'FDHH_Sr25.wav' 'FDHH_Sr26.wav' '
    FEAB_Sr5.wav' 'FEAB_Sr6.wav' ...
'FHRO_Sr31.wav' 'FHRO_Sr32.wav' 'FJAZ_Sr35.wav' 'FJAZ_Sr36.wav' 'FMEL_Sr21.
    wav' 'FMEL_Sr22.wav' ...
'FMEV_Sr10.wav' 'FMEV_Sr9.wav' 'FSLJ_Sr33.wav' 'FSLJ_Sr34.wav' 'FTEJ_Sr13.
    wav' 'FTEJ_Sr14.wav' ...
'FUAN_Sr39.wav' 'FUAN_Sr40.wav' 'MASM_Sr11.wav' 'MASM_Sr12.wav' 'MCBR_Sr23.
    wav' 'MCBR_Sr24.wav' ...
'MFKC_Sr43.wav' 'MFKC_Sr44.wav' 'MKBP_Sr19.wav' 'MKBP_Sr20.wav' 'MLKH_Sr37.
    wav' 'MLKH_Sr38.wav' ...
'MMLP_Sr27.wav' 'MMLP_Sr28.wav' 'MMNA_Sr15.wav' 'MMNA_Sr16.wav' 'MNHP_Sr1.
    wav' 'MNHP_Sr2.wav' ...
'MOEW_Sr41.wav' 'MOEW_Sr42.wav' 'MPRA_Sr29.wav' 'MPRA_Sr30.wav' 'MREM_Sr7.
    wav' 'MREM_Sr8.wav' ...
'MTLS_Sr17.wav' 'MTLS_Sr18.wav'};


train = cell(length(initial),length(sentence));

for i=1:length(initial)
    for s=1:length(sentence)
       temp = [traindir initial(i,:) '_S' sentence(s) '.wav'];
       tempwav = wavread(temp);
       train{i,s} = tempwav;
    end
end

test = cell(length(initial),2);

for i=1:length(initial)
   for s=1:2
      temp = [testdir filename{(i-1)*2+s}];
      tempwav = wavread(temp);
      test{i,s} = tempwav;
   end
end
```

A.4 computeweights.m

```
function w=computeweights(codebooks)

for i=1:length(codebooks)           % loop over all codebooks
    for j=1:size(codebooks{1},1)    % loop over all codevectors

        s=0;
        for k=1:length(codebooks)   % find nearest codevector from all other
            codebooks
            if k~=i                 % codebooks must be different
                dmin = min(disteusq(codebooks{i}(j,:), codebooks{k}, 'x'));
                s = s + 1/dmin;
            end
        end
        w(i,j) = 1/s;

    end
end
```

A.5 cc.m

```
function y = hmmfeatures(s,N,deltaN,M,Q)
%  hmmfeatures --> Feature extraction for HMM recognizer.
%
%  <Synopsis>
%    y = hmmfeatures(s,N,deltaN,M,Q)
%
%  <Description>
%    A frame based analysis of the speech signal, s, is performed to
%    give observation vectors (columns of y), which can be used to train
%    HMMs for speech recognition.
%
%    The speech signal is blocked into frames of N samples, and
%    consecutive frames are spaced deltaN samples apart. Each frame is
%    multiplied by an N-sample Hamming window, and Mth-order LP analysis
%    is performed. The LPC coefficients are then converted to Q cepstral
%    coefficients, which are weighted by a raised sine window. The result
%    is the first half of an observation vector, the second half is the
%    differenced cepstral coefficients used to add dynamic information.
%    Thus, the returned argument y is an 2Q-by-T matrix, where T is the
%    number of frames.
%
%  <See Also>
%    hmmcodebook --> Codebook generation for HMM recognizer.

%  <References>
%  [1] J.R Deller, J.G. Proakis and F.H.L. Hansen, "Discrete-Time
%      Processing of Speech Signals", IEEE Press, chapter 12, (2000).
%
%  <Revision>
%    Peter S.K. Hansen, IMM, Technical University of Denmark
%
%    Last revised: September 30, 2000
%-------------------------------------------------------------------

Ns = length(s);                          % Signal length.
T  = 1 + fix((Ns-N)/deltaN);             % No. of frames.

a         = zeros(Q,1);
gamma     = zeros(Q,1);
gamma_w = zeros(Q,T);

win_gamma = 1 + (Q/2)*sin(pi/Q*(1:Q)'); % Cepstral window function.

for (t = 1:T)                            % Loop frames.
  % Block into frames.
  idx = (deltaN*(t-1)+1):(deltaN*(t-1)+N);

  % Window frame.
  sw = s(idx).*hamming(N);

  % Short-term autocorrelation.
  [rs,eta] = xcorr(sw,M,'biased');

  % LP analysis based on Levinson-Durbin recursion.
  [a(1:M),xi,kappa] = durbin(rs(M+1:2*M+1),M);

  % Cepstral coefficients.
  gamma(1) = a(1);
  for (i = 2:Q)
    gamma(i) = a(i) + (1:i-1)*(gamma(1:i-1).*a(i-1:-1:1))/i;
```

```
    end

    % Weighted cepstral sequence for frame t.
    gamma_w(:,t) = gamma.*win_gamma;
end

% Time differenced weighted cepstral sequence.
delta_gamma_w = gradient(gamma_w);

% Observation vectors.
y = [gamma_w; delta_gamma_w];

%---------------------------------------------------------------------------
% End of function hmmfeatures
%---------------------------------------------------------------------------
```

A.6 durbin.m

```
function [a,xi,kappa] = durbin(r,M)
%  durbin --> Levinson-Durbin Recursion.
%
%  <Synopsis>
%    [a,xi,kappa] = durbin(r,M)
%
%  <Description>
%    The function solves the Toeplitz system of equations
%
%       [  r(1)   r(2)  ...  r(M)  ] [  a(1)  ] = [  r(2)  ]
%       [  r(2)   r(1)  ... r(M-1) ] [  a(2)  ] = [  r(3)  ]
%       [   .      .           .   ] [   .    ] = [   .    ]
%       [ r(M-1) r(M-2) ...  r(2)  ] [ a(M-1) ] = [  r(M)  ]
%       [  r(M)  r(M-1) ...  r(1)  ] [  a(M)  ] = [ r(M+1) ]
%
%    (also known as the Yule-Walker AR equations) using the Levinson-
%    Durbin recursion. Input r is a vector of autocorrelation
%    coefficients with lag 0 as the first element. M is the order of
%    the recursion.
%
%    The output arguments are the M estimated LP parameters in the
%    column vector a, i.e., the AR coefficients are given by [1; -a].
%    The prediction error energies for the 0th-order to the Mth-order
%    solution are returned in the vector xi, and the M estimated
%    reflection coefficients in the vector kappa.
%
%    Since kappa is computed internally while computing the AR coefficients,
%    then returning kappa simultaneously is more efficient than converting
%    vector a to kappa afterwards.
%
%  <See Also>
%    rf2lpc --> Convert reflection coefficients to prediction polynomial.
%    lpc2rf --> Convert prediction polynomial to reflection coefficients.

%  <References>
%  [1] J.R Deller, J.G. Proakis and F.H.L. Hansen, "Discrete-Time
%      Processing of Speech Signals", IEEE Press, p. 300, (2000).
%
%  <Revision>
%    Peter S.K. Hansen, IMM, Technical University of Denmark
%
%    Last revised: September 30, 2000
%-------------------------------------------------------------------------

% Initialization.
kappa = zeros(M,1);
a     = zeros(M,1);
xi    = [r(1); zeros(M,1)];

% Recursion.
for (j=1:M)
  kappa(j) = (r(j+1) - a(1:j-1)'*r(j:-1:2))/xi(j);
  a(j)     = kappa(j);
  a(1:j-1) = a(1:j-1) - kappa(j)*a(j-1:-1:1);
  xi(j+1)  = xi(j)*(1 - kappa(j)^2);
end
%-------------------------------------------------------------------------
% End of function durbin
%-------------------------------------------------------------------------
```