EEC 201 Final Project:
Speaker Recognition System

Winter 2020 By Chen Lu and Rongfei Li

Objective

Our main goal is to build an automatic speaker recognition system with MATLAB as well as a graphical user interface (GUI) for the program. On top of that, to carry out exhaustive experiments to test and analyze the performance of the speaker recognition system.

Overview

Speaker recognition is the process of automatically recognizing the speaker based on the information included in speech waves. Speaker recognition can be further classified into identification and verification. Speaker identification is the process of distinguishing the speaker on the basis of registered voice. Speaker verification, on the other hand, is the process of accepting or rejecting the identity claim of a speaker. The principles and implementations of two types are quite distinguishing, while it won't take much work to convert from one to another as we shall discuss later.

In this project we will be mainly focusing on the implementation of the speaker identification system. A diagram depicting the system is given as below.

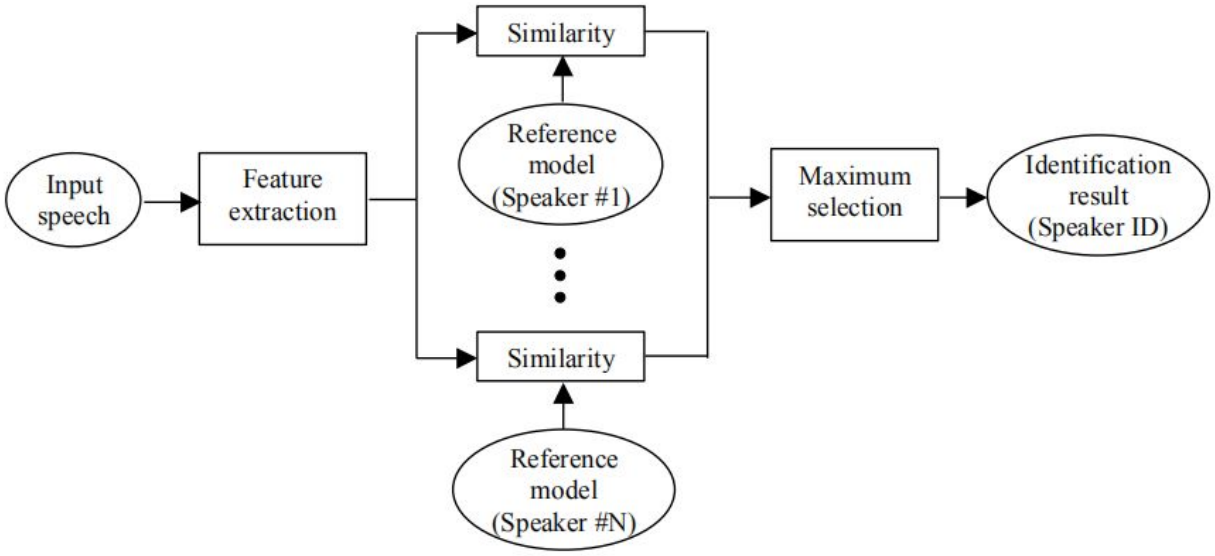

Figure 1. Structure of speaker identification system

The speaker recognition system is divided into two distinguished phases. The first one is referred to the training phase, while the second one is the testing phase.

In the training phase, each registered speaker has to provide samples of their voice by uttering the same word ‘zero’ so that the system can build a feature model for the speaker. In the testing phase, the testing voice signal is matched with the stored reference model and a recognition is made accordingly.

We used the Mel-Frequency Cepstrum Coefficients (MFCC) method to represent the features of each speaker with an unique codebook. The Vector Quantization (VQ) method was then implemented to match the testing voice with the stored samples. The matching decision was made by calculating the Euclidean distance between the codebook vector and the testing set vector.

Design and Test Procedure

Speech Data

Before diving into the design of each signal processing function, we played each audio file in the provided training set and testing set. The provided test sets are from distinguished people who utter the same word ‘zero’. Figure 2 displays three different sets of .wav file in the time domain.

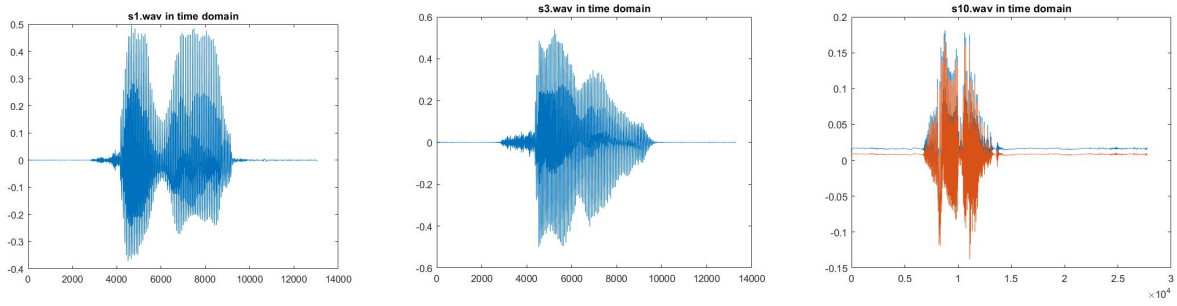

Figure 2. Raw speech signal in time domain from three different speakers

By inspecting the audio waveforms, it is not hard to find that there is a huge amount of data for even a small block of waveform, which makes it hard to analyze the characteristics of each speaker directly in time domain.

Besides, although each speaker is uttering the same word, we notice that they are not synchronized and have different lengths of sound.

Furthermore, each training audio signal has different power, even different channel number. All mentioned above may bring unexpected effects to our processing progress, which motivate us to preprocess the original data before implementing the feature extraction.

Speech Processing (MFCC)

Figure 3 demonstrates the procedures we implemented in the speech processing phase.

Firstly, the mono signal was normalized by the amplitude of the training data to force it into the range of -1 and 1.

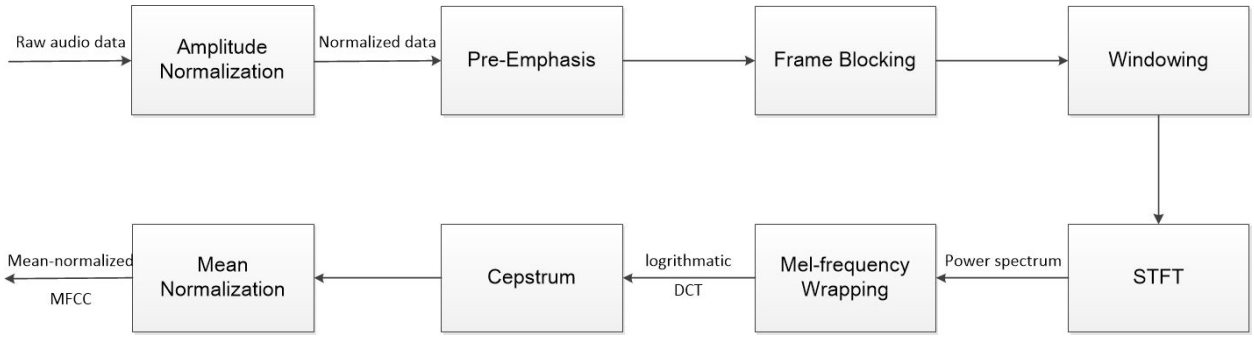

Figure 3. Block diagram of the speech processing procedure

The next step was to apply a pre-emphasis filter on the signal to amplify the high frequency components, as shown in the code below.

```
%pre-emphasis
b=[1 -0.95];
H=freqz(b,1,'whole',length(s));
S=fft(s);
ES=S.*H;
s=ifft(ES);
```

As we can tell from Figure 4 and 5, the pre-emphasis filter balanced the frequency spectrum since high frequency components usually have smaller magnitudes compared to lower frequencies.

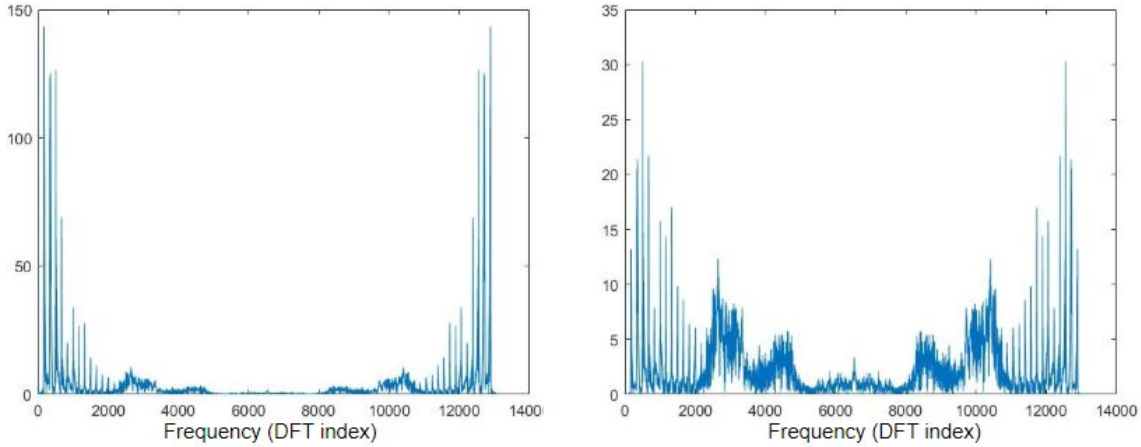

Figure 4. The frequency response before pre-emphasis (left) versus after pre-emphasis

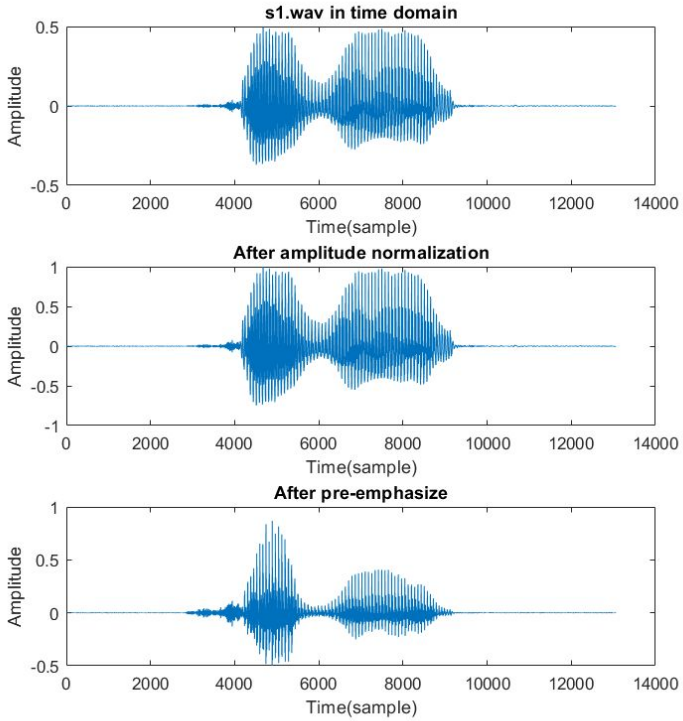

Figure 5. Preprocessing the audio data displayed in time domain

After pre-emphasis, we split the signal into short-time frames and added Hamming windows to counter the spectrum leakage.

```
%frame blocking
Nframes = floor(( length(s) - n) / m) + 1;%number of total frames
for i = 1:n
   for j = 1:Nframes
      F(i, j) = s(((j - 1) * m) + i); %F(i,j) is the ith element in the jth frame
   end
end
%windowing
w = hamming(n);%w is the hamming window
Fw = diag(w) * F;%implement windowing
```

Short time Fourier Transform (STFT) was then carried out to find the spectrogram of the training signals, as shown in Figure 6.

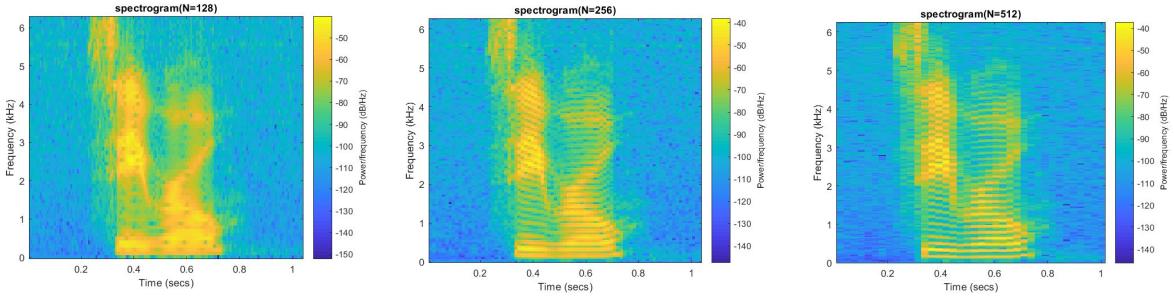

Figure 6. Spectrograms with different frame length

As we increase the frame size, we get better resolution in frequency domain while worse resolution in time domain. Therefore we choose an intermediate value N = 256 for the better performance.

The power spectrum (periodogram) was then obtained using the following equation:

$$P = \frac{|FFT(x_i)|^2}{N}$$

This can be implemented simply by taking the square of FFT magnitude and dividing N, which is 256 in this scenario. The result spectrum is given as Figure 7.

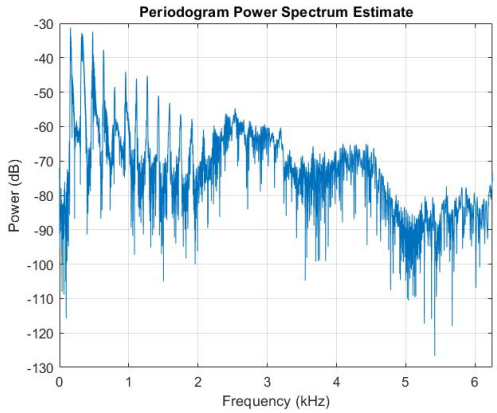

Figure 7. Spectrum (Periodogram) of s1.wav after preprocessing

In the next step, we generated the Mel-scale filter banks. The filter banks aim to mimic the non-linear human ear perception of sound, by being more discriminative at lower frequencies and less discriminative at higher frequencies. MATLAB script FilterBank.m was built to generate the Mel-scale filter banks as shown below.

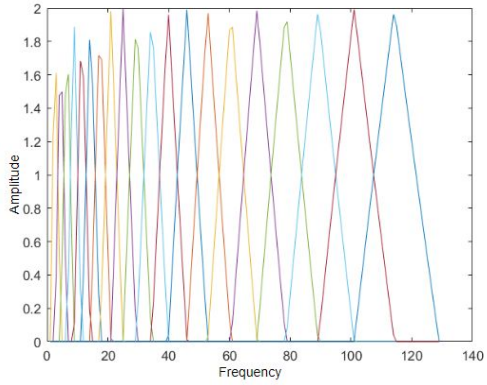

Figure 8. Filter bank on Mel-Scale

The filter banks obtained in the preceding step were then applied to the log spectrum, resulting in the cepstrum. Because the mel spectrum coefficients are real, we converted them to the time domain using the Discrete Cosine Transform (DCT). The DCT was needed to decorrelate filter bank coefficients. The Mel-Frequency Cepstrum Coefficients were then given by:

$$c_n = \sum_{k=1}^{K} (\log S_k) \cos\left[n\left(k - \frac{1}{2}\right)\frac{\pi}{K}\right], \qquad n = 0, 1, \ldots, K\text{-}1$$

The resulting cepstral coefficients 2 - 13 were retained and the rest were discarded because they represent fast changes in the filter bank coefficients and these fine details do not contribute much to our recognition procedure.

```
mf=mf-mean(mf,2)+1e-8;%mean normalization
c = dct(log(mf));%cepstrum
c=real(c(2:13,:));
c=c-mean(c,2)+1e-8;%mean normalization
```

Before obtaining the MFCC coefficient outputs, we also normalized out the mean values in each filter bank and MFCC so that the spectrum got balanced and better SNR. Part of the resulting MFCC's are visualized as shown in Figure 9 and 10.

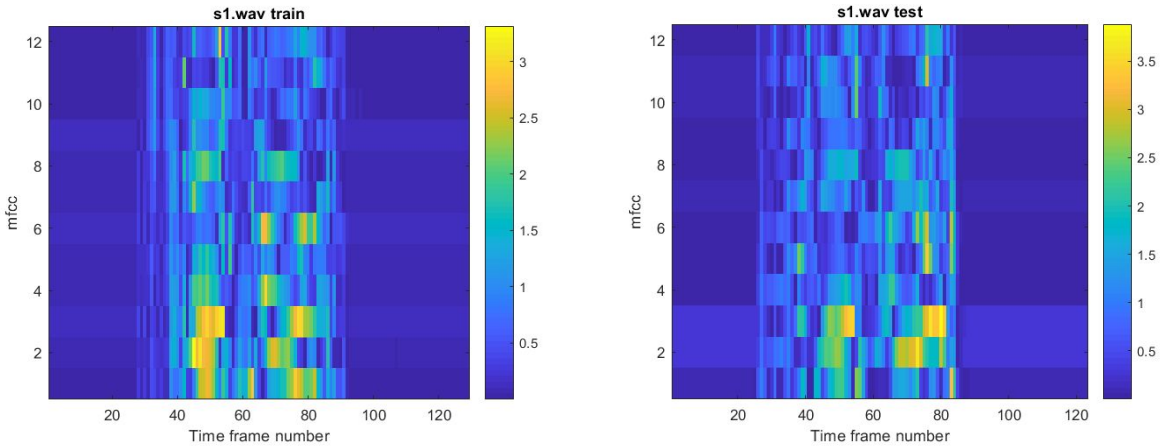

Figure 9. MFCC results of s1.wav in training set (left) and testing set (right)

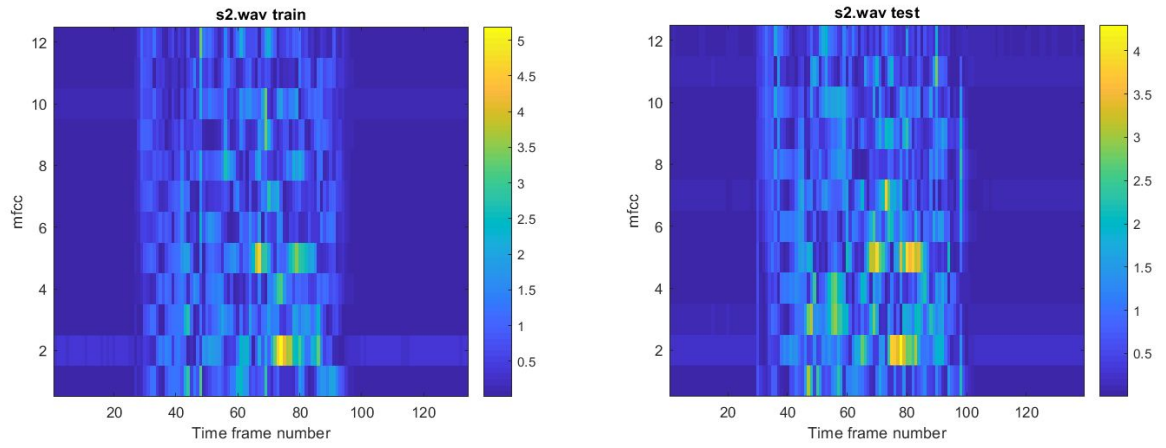

Figure 10. MFCC results of s2.wav in training set (left) and testing set (right)

By inspecting the MFCC's of different speakers, we can tell distinct differences both in time domain and MFCC coefficients, i.e. frequency domain. On the other hand, there are clear common features in the MFCC's from the same speaker. In a word, it makes sense to represent the characteristics of different speakers with the MFCC coefficients.

Vector Quantization

After processing the speech signal, we have to further consider how to classify the input signal with the registered categories or classes. In this project, the Vector Quantization (VQ) method was used, due to ease of implementation and high accuracy.

VQ is a process of mapping vectors from a large vector space to a finite number of regions in that space. Each region is called a cluster and each cluster can be represented by its centroid, or codeword. The collection of all codewords of one category (tester, in this case) is called a codebook.

The LBG (Linde, Buzo and Gray) algorithm was implemented in this project. The algorithm was implemented by the recursive procedure described in Figure 11. The MATLAB function VQ.m was built to implement the algorithm.

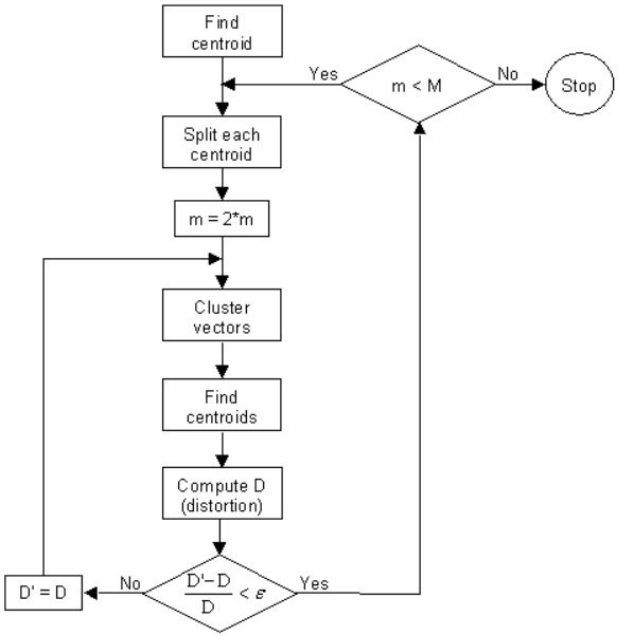

Figure 11. Flow diagram of the LBG algorithm (Rabiner and Juang, 1993)

The distortion is calculated by the Euclidean distance between two vectors:

$$D(x, y_i) = \sqrt{\sum_{j=1}^{k} (x_j - y_{ij})^2}$$

After quantization, we chose two dimensions in the VQ codewords and made a comparison with the MFCC vectors, as shown in Figure 12. Each symbol with different color represents a disparate speaker.

As we can tell, a quite large portion of symbols are overlapping with others. However, in some areas the phenomenon of clustering is obvious. For instance, the up portion of the graph is mainly occupied by the

blue stars. Such clustering behavior in each dimension of the VQ vector allows us to model the features of each speaker, so that we can identify the unknown voice that shall be discussed later.

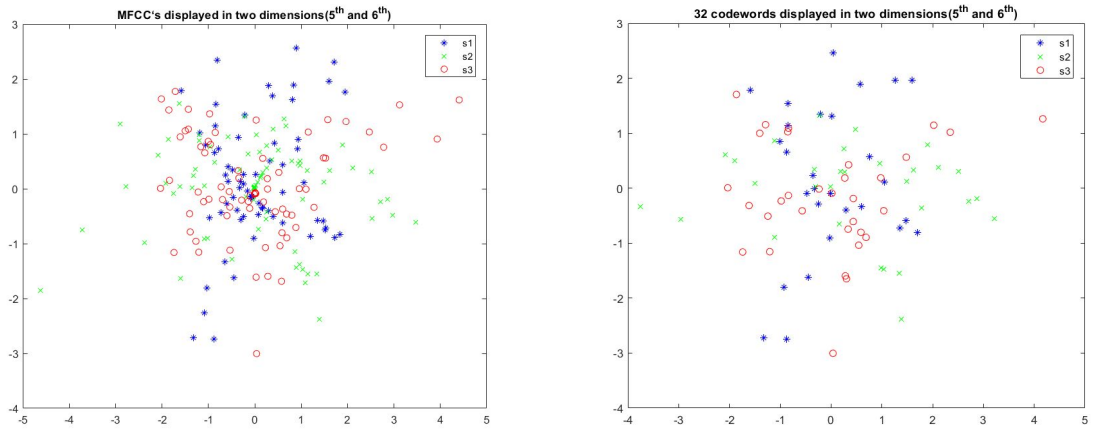

Figure 12. Fifth and sixth row of the MFCC's and 32-VQ codewords vector

In the recognition phase, an input utterance of an unknown voice is vector quantized using each trained codebook and the total distortion is computed. The speaker corresponding to the VQ codebook with smallest total distortion is identified as the speaker of the input utterance.

Test Results and Analysis

To test how well is our Speech Processing and Vector Quantization process, we generated 22 audio files that 11 distinct speakers say the same word 'zero' once in each file. So each speaker said the word twice in total and the audios have been saved in separate files. One audio file by each speaker is saved as 'training.wav', which will be used in the training of the Vector Quantization process. The other audio file is saved as 'testing.wav', which will be feeded in as a testing file and to check the accuracy of our model. So there are totally 11 files for training and 11 files for testing.

Primitive Test Results

In this section, we will summarize the testing results of each speaker without contaminating noise in the samples. The result is shown in the following table. On the left column is the testing sample number to be tested and on the right column is the codebook number that has been matched with the testing sample.

Table1 shows the result that only testing sample 7 is mismatched with Codebook 4. If we calculate the accuracy = $\frac{\textit{sample numbers that matches correctly}}{\textit{total numbers of samples}} = \frac{10}{11} = 90.9\%$. This number shows that our algorithm can identify speakers very well without adding noises to the signal.

Speaker testing sample number	Codebook number that matches the sample
1	1
2	2
3	3
4	4
5	5
6	6
7	4
8	8
9	9
10	10
11	11

Table1. Primitive test results

Noise Sensitivity Test

In this section, we add different levels of random noise to the testing signal and to check how well the system can identify the speaker. Since the magnitude of our speech signals have been normalized between -1 and 1, adding a random noise between -x to x to the signal means adding a x% noise to the samples. Where x is taken between 0 and 1. The sensitivity plot where shows the accuracy against noise level has been shown below.

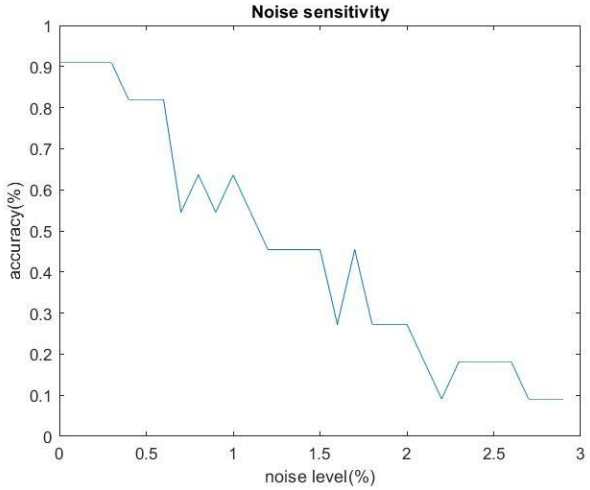

Figure 13. Sensitivity plot (noise level versus accuracy)

Figure 13 shows that as noise level increases, the accuracy decreases and reduces as close as to 10% when noise level approaches 3%. Therefore, our system requires noises to be below a certain level to have reasonably good performance.

User Identification Test

In this section, we create a GUI to register two users' voices and compare them to original registered voices in the database. We found out that users have to speak in some special ways to identify between users with similar voices. For example, in one of our testing scenarios, one user has to emphasize the pronunciation of 'Z' to identify himself with other users. Since our data has been normalized, the magnitude of voices don't affect the accuracy, which is tested by speaking from different distances to the system(computer).

Also, we found out the accuracy is also related to the centroids number created for each codebook. If the centroids number is too small, accuracy will drop since MFCC vectors cannot be split clearly because of large overlaps of each cluster. However, if entroids number is too large, accuracy will also drop because of the oversampling of training data; The clusters will emphasize more on the signal difference from noise instead of difference from features.

Conclusions and Future works

Our speaker recognition system shows fairly good performance to identify different speakers if the samples are not contaminated by significant noises. As the noise level increases, the performances of our system drops rapidly. The number of filter banks certainly have effects on the performance as the number of filter banks specifies the resolution of each frequency projecting to the cepstrum. Future work could include the research on varying the number of filter banks and find the optimal value of this number.

References

1. Final Project Manual.
2. https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html
3. F.K. Song, A.E. Rosenberg and B.H. Juang, "A vector quantisation approach to speaker recognition", AT&T Technical Journal, Vol. 66-2, pp. 14-26, March 1987.
4. Y. Linde, A. Buzo & R. Gray, "An algorithm for vector quantizer design", IEEE Transactions on Communications, Vol. 28, pp.84-95, 1980.
5. http://minhdo.ece.illinois.edu/teaching/speaker_recognition/speaker_recognition.html
6. Tejal Chauhan, Hemant Soni, Sameena Zafar, A Review of Automatic Speaker Recognition System, International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-3, Issue-4, September 2013.