Final Project

Speaker Recognition

EEC 201: Digital Signal Processing

Professor: Zhi Ding

Students: Yuyang Jin, Joey Koury

March 18, 2024

Introduction

For our final project, our task was to create a speaker recognition system that would automatically recognize who is speaking on the basis of individual information included in speech waves. The main idea is to have a training set of data, with numerous people all saying the same word. Additionally, we would have another set of testing data, with those same people saying the exact same word a second time. Our system would use the training data set as an initial input, creating a codebook for each of the speakers utilizing feature extraction. Next, we would input the testing data into our system, and try to match each speaker from the training set to the testing set using feature matching.

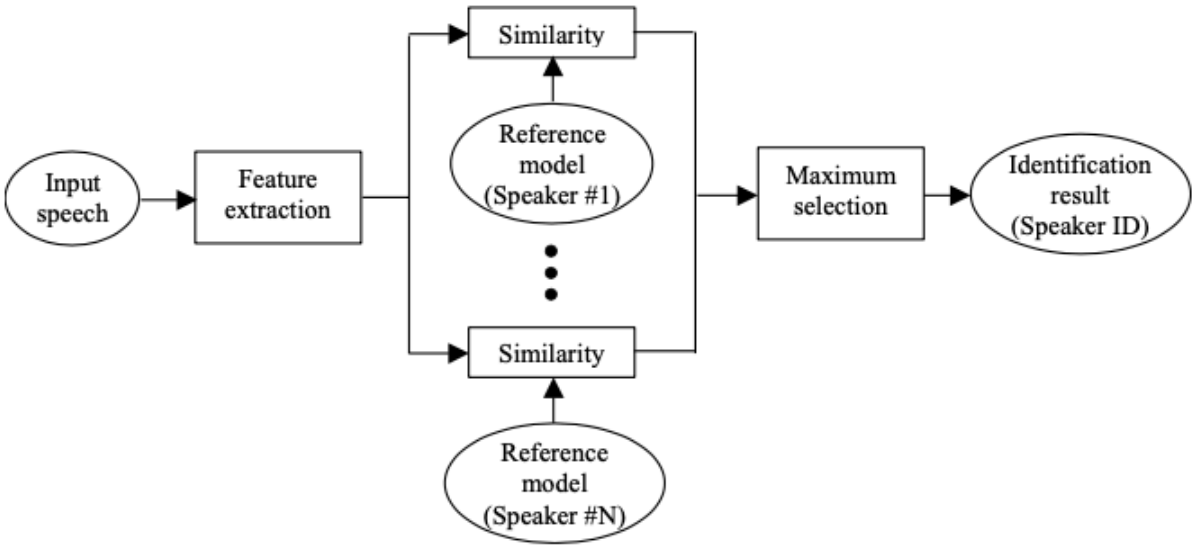

Figure 1: Basic structure of speaker identification systems

Procedure

For our test, we had a group of people recite the words “zero” and “twelve” twice each. The first step was to find the Mel-frequency cepstrum coefficients (MFCC) of each training set. “Zero” was the word that each person said. To do this, we input the recordings into Matlab, and followed the steps outlined in figure 2 to reach our MFCC coefficients.

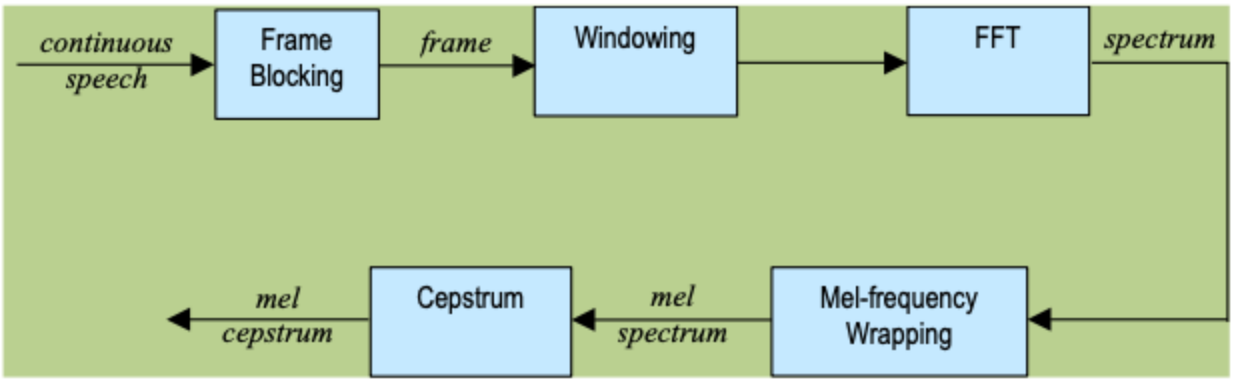

Figure 2: Block diagram of the MFCC processor

We first blocked each recording into a set number of frames, Lu. Each frame had N number of samples. Typically, N is about 256, but this is a variable which can vary to find the optimal speech recognition system. Next, for every frame, we split it into K=20 bins. Furthermore, we implemented a hamming window over each frame.

Next, we take the FFT of each windowed data, and apply weights calculated using mel-spaced filterbank (see figure 3).

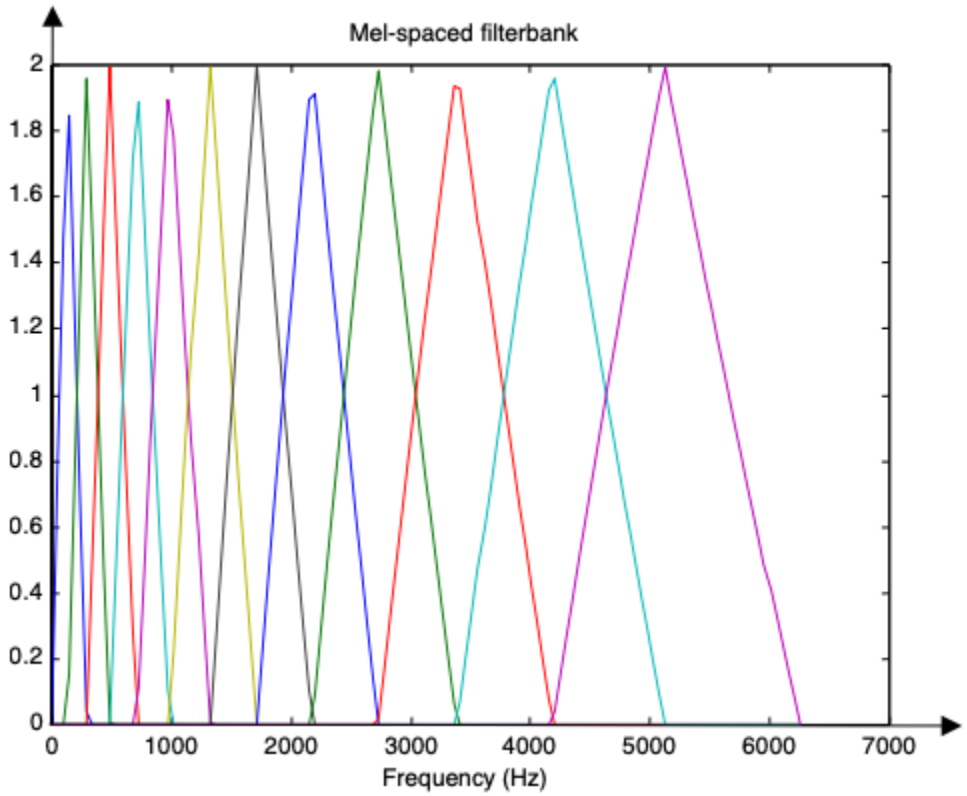

Figure 3: Mel-spaced filterbank example

Finally, we take the cepstrum, which converts back from frequency domain to time domain using discrete cosine transform, and then calculates the weighted sum for each bin. Hence, we are left with a KxLu matrix which contains the mel frequency cepstrum coefficients (MFCC) coefficients for each person saying "zero". We decided to drop the first component of the MFCC coefficients, c_0 as it represents the mean value of the input signal and contains little speaker specific information.

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

Equation 1: MFCC coefficient equation

Finally, we will be left with a 19xLu matrix for each person speaking their word.

Next, our task was to take all Lu data points from each MFCC matrix and find centroids. Therefore, if there were roughly $Lu \approx 128$ data points, we would try to cluster those data points and create 16 or 32 centroids that would represent those data points. To do this it involved writing an LBG algorithm which iterated continuously until high quality centroids were found that were at a minimum distance between itself and the data points. To have a visual of this, imagine if we were just dealing with 2 dimensions, so we had a 2xLu matrix.

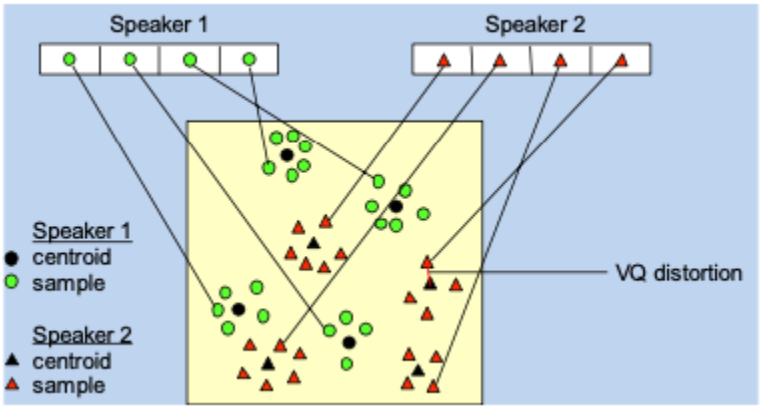

Figure 4: Illustration of creating centroids from training dataset

Figure 4 shows a 2D visual representation of how one would cluster the data sets for a couple different speakers and find their centroids. The method is to first create one centroid which is the mean of all the data points for that particular speaker. Next, split that centroid into two, with the centroid locations being at $(1 + \varepsilon)y_n$ and $(1 - \varepsilon)y_n$, where y_n is the location of the initial centroid, and epsilon being ~0.01. We then keep updating the location of the centroid until a distance requirement of $\frac{D_{n-1} - D_n}{D_n} < 0.001$ is met, where D_n is the mean minimum distances

between data points' closest centroid and itself. Once the distance criterion is met, the two centroids will split into 4, and the process repeats itself. This process will repeat until the number of centroids has met our predetermined number of centroids (usually around 32 centroids).

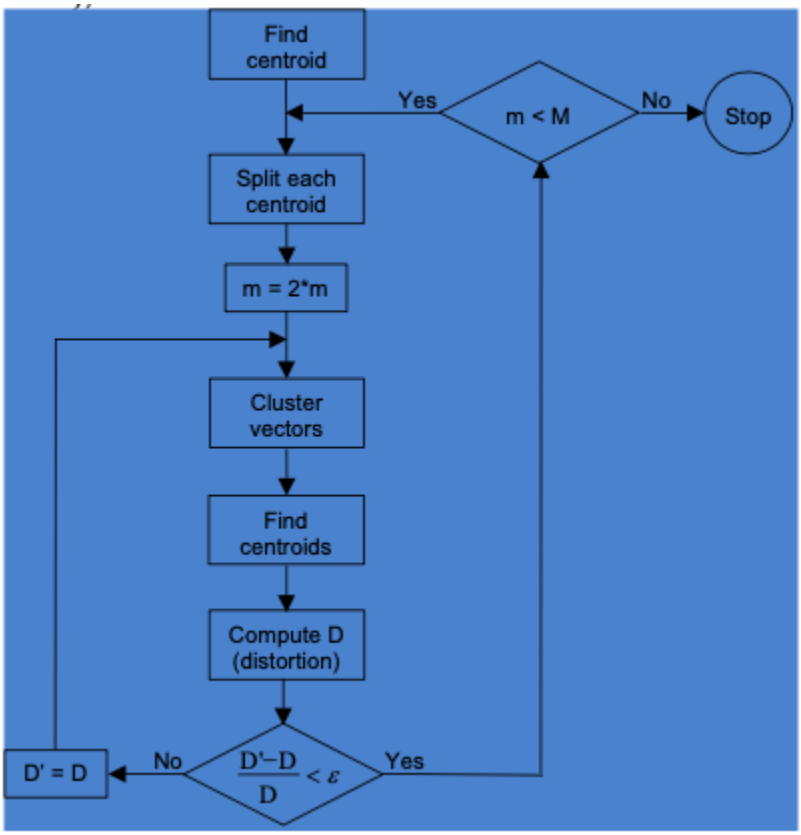

Figure 5: Flowchart of LBG algorithm

Lastly, after finding centroids for each speaker, we find MFCC coefficients for the testing data set, and match those coefficients with the centroids. Whichever match has the smallest minimum distance is our winner!

Data

If we take one person's training data, such as given speech data person 3, we can use them as an example to show how we find MFCC coefficients.

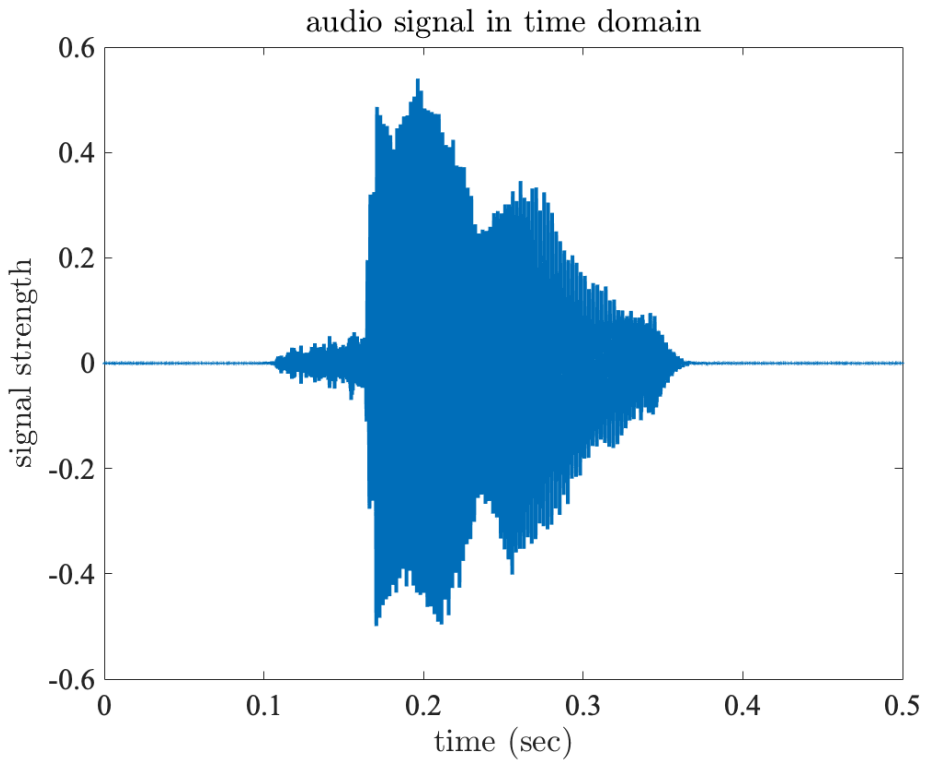

Figure 6: Example of plotting an audio signal in time domain

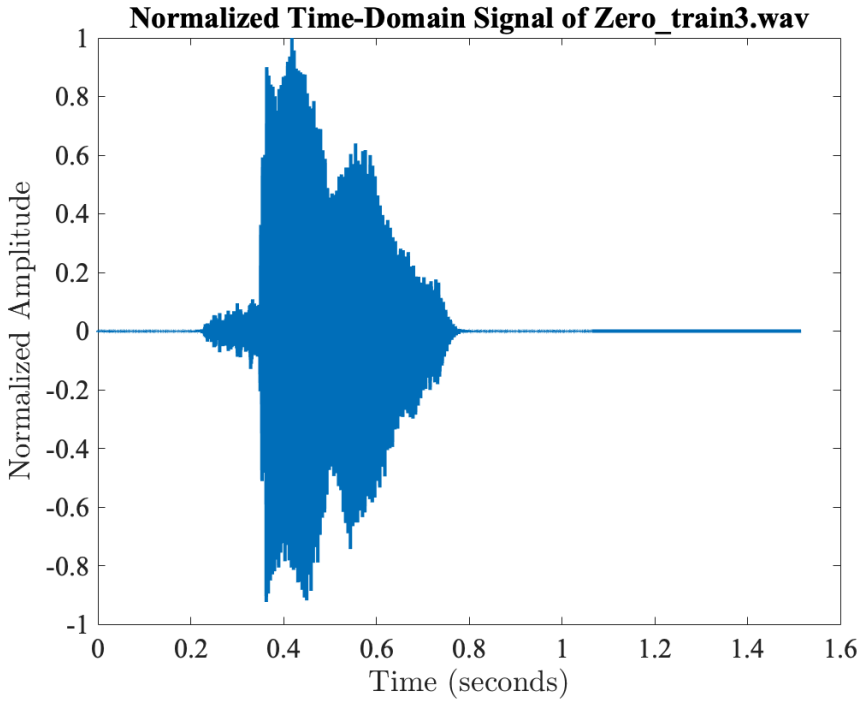

Figure 7: Example of audio signal in time domain after normalization

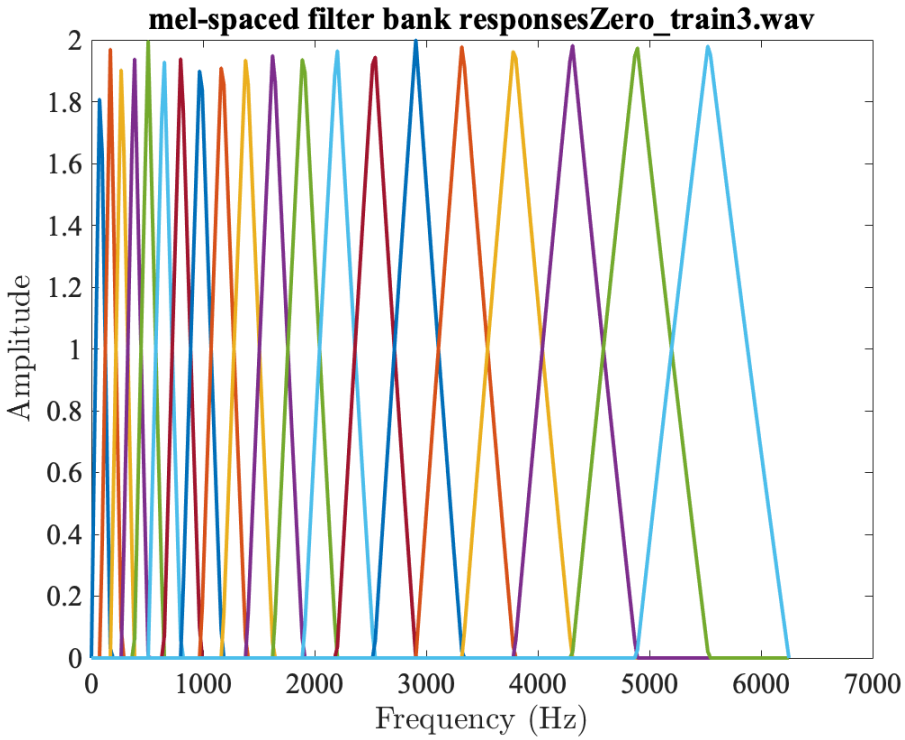

Figure 8: Mel-Space Filter Bank Response

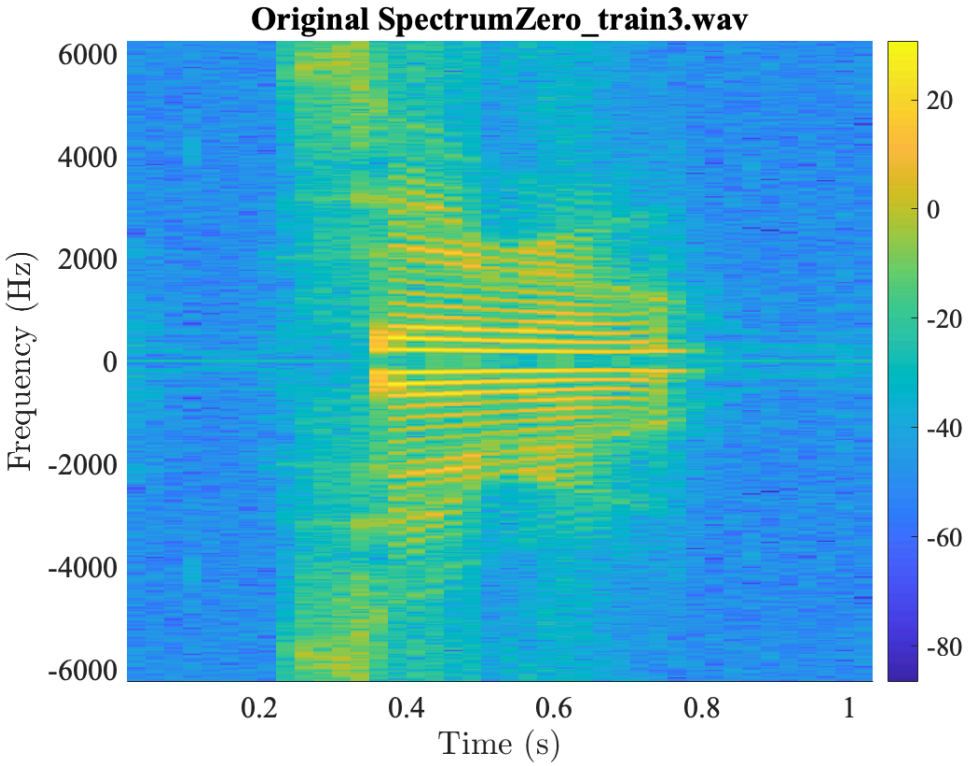

Figure 9: Spectrum of speech file before mel-frequency wrapping

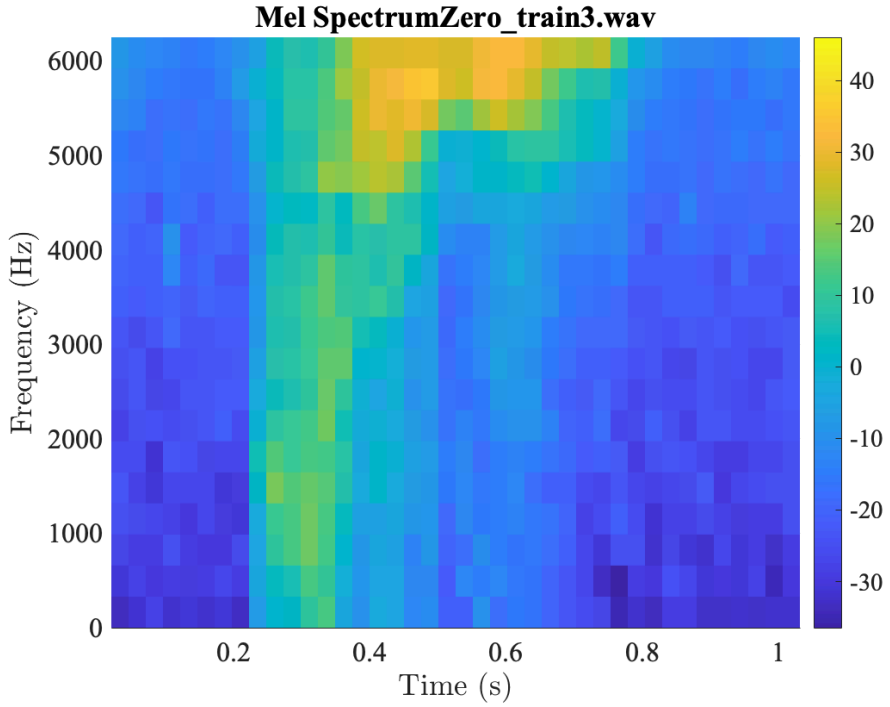

Figure 10: Spectrum of speech file after mel-frequency wrapping

Psychophysical studies have discovered that human perception of sound frequency does not follow a linear scale. Therefore, we use the melfb function to scale frequencies. The melfb function takes the number of desired Mel filters(K), the FFT length (n), and the sampling rate (fs) as inputs and generates a matrix containing K filters to represent the Mel filter's responses. We can apply these filters to FFT results to generate spectrums that emphasize frequencies in a way that aligns with how people perceive sound. Therefore, it is very useful to apply the weighted logarithms onto the signal.

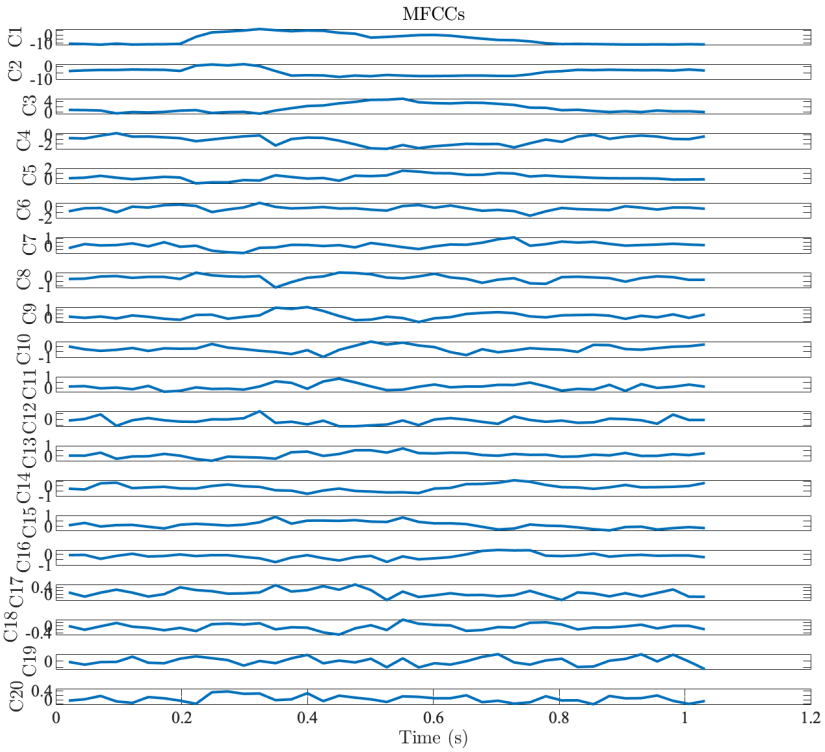

Figure 11: Visual representation of each c_n coefficient in the MFCC coefficient matrix.

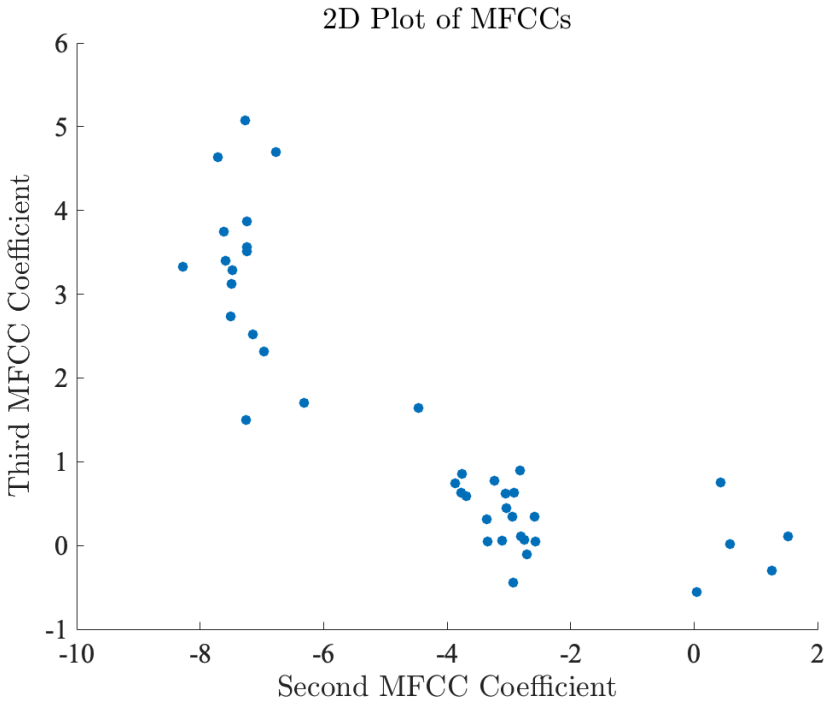

Figure 12: 2D illustration of data points.

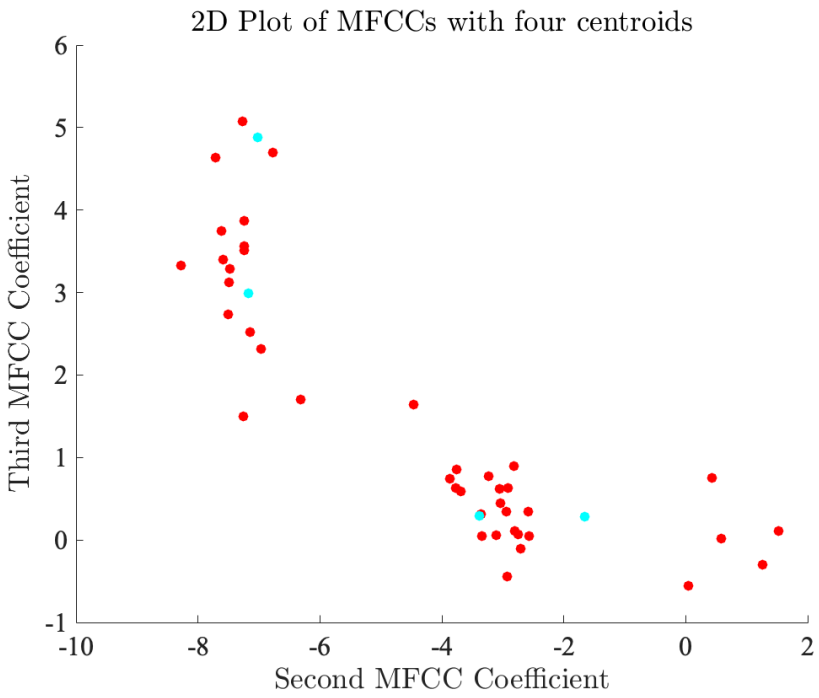

Figure 13: Example of where centroids would be for a 2D matrix with four centroids

Results

Our first task was to test our program against a given set of speech data. There were 11 training audio files, and we were to test it with 8 testing audio files. Using our own ear, we found that test file 1 matched with training file 1, test file 2 matched with training file 2, and so on until test file 8 matched with training file 8. We were confident with 6/8 that the trend of training file n = testing file n, but for a couple weren't 100% positive. Hence, we can say that the human ear can perform voice recognition with a 75% success rate. We were able to run our code and successfully identify all 8 speakers with a 100% success rate. This required no mettling with our code, as it worked the first time using 16 centroids and our default of N = 256.

Additionally, we added in a notch filter. Our system has a sampling frequency of 48kHz. Therefore, we tested the robustness of our system by applying a notch filter with a stopband from 5.0kHz to 5.1kHz. With the bandstop filter killing 100Hz of our data, we were still able to get a 8/8, 100% success rate. However, when we increased our stop band to range from 5.0kHz to 5.5kHz, our success rate drops to 4/8, 50%.

Next, we began using human audio recordings from our classmates in our data. We found that when we used 10 different people saying "zero", as well as the one person saying "zero", "yes", and "no", we were able to match 11/13 correctly. The two that weren't matched correctly were in the dataset amongst our classmates. On the other hand, the three additional signals that were

added on all matched correctly. This further highlights the robustness of our system, as it can handle other words beyond just numbers.

Furthermore, we tested our entire class of 18 people saying "zero" and "twelve." We found that for "zero," our best success rate was 14/18, or 78%. Similarly, for "twelve," our best success rate for 14/18.

Discussion

Our biggest struggle in completing this project was getting our LBG algorithm to work. Yuyang was able to pretty quickly develop the code to find the MFCC coefficients; however, Joey took a while to write the algorithm to find centroids. To confirm our issues were in the LBG algorithm and not somewhere else, we used kmeans (matlab's built in clustering algorithm) to find the centroids. We discovered that when we used kmeans, our code worked well and we achieved a 6/8 success rate on the given speech data. Hence, we were fairly confident that our MFCC matrix was being produced correctly, but our LBG algorithm was lacking. The first problem was that Joey had a misunderstanding of how to develop the centroids. At first, we were placing our final number of centroids in the 19 dimensional space, and then iterating to refine the position of the centroids. However, after discussing with more classmates, we learned the better approach was to create a couple centroids first, refine the position of those two, and then split into four centroids. By repeatedly splitting and refining positions until the desired number centroids have been met, we were able to get better results. While the first clustering algorithm – which immediately jumped to the final number of desired centroids – worked 100% on the given speech data, it was only achieving a 50% success rate on the classroom recorded data. On the other hand, when using the improved clustering algorithm, we got to a 78% success rate on the classroom data.

To try and improve our system, we placed our code in a nested for loop, where we tried different frame sizes (and hence different overlaps), as well as different number of centroids. Unfortunately, after letting the system run for hours testing different possibilities, we weren't able to go beyond a 14/18 success rate. We did find that going from N=256 to N=512 did increase our success rate from 12/18 to 13/18, but going to N=1024 did not help. Furthermore, on one hand, we could have kept increasing the number of centroids, as going from 32 to 64 centroids did improve our success rate from 13/18 to 14/18. However, we didn't want to go beyond 64 centroids, as at a certain point, increasing centroids would basically render the act of clustering moot.

Moreover, another interesting finding was that using energy normalization decreased our success rate. Normalization is used to take into account how some people might speak louder or quieter

than others. Furthermore, it can also help with different people speaking faster or slower. Therefore, with normalization in place, if a particular person speaks loudly during training, but softer in testing, the system would still be able to match them. However, it makes sense that if someone spoke softly during the training set, they would speak softly again during testing. Therefore, by omitting energy normalization, our system had more differences between each speaker to juxtapose, essentially having more data to work with, causing our success rate to be higher. I would imagine if we conducted our training and testing audio recordings in different environments, such as one inside the classroom and another at an amusement park, using energy normalization would be beneficial.

Conclusion

Overall this project was a success, as we were able to get our speaker recognition system working with a 100% success rate with the given speech data, and a high percentage on the classroom data. Also, it was very nice that this was a partner project, as we often caught each other's typos in code which otherwise would have caused days and days of headache. Next time, however, I believe we could have better teamwork, as we both did a lot of work that overlapped onto each other. Moreover, we also struggled with a language barrier, as Joey's Chinese was limiting, and so was Yuyang's English.

Yuyang mostly took charge of creating the MFCC coefficient matrix – in particular tests 1-4. Joey took responsibility for creating the LBG algorithm – in specific tests 5-6. After the initial excitement of the code working on test 7, both Yuyang and Joey got a second wind on the project, and simultaneously did tests 9 and 10. Yuyang also worked on creating the notch filter, test 8. Lastly, Yuyang and Joey both worked on the written report as well as the video recap.

Video Summary Link
https://youtu.be/ZS9sSn07KBQ

Github Link
https://github.com/yuyjin/Audio-Masters/