Speaker Recognition

University of California, Davis

EEC 201: Digital Signal Processing Winter Quarter 2021

Professor Zhi Ding

Authors: Trevor Vidano, Tanuj Kalakuntla

Introduction

The purpose of this project is to build a speech recognition model that would be able to recognize a specific speaker based on specific characteristics of the individual. Applications of this relate to using an individual's voice for security or authentication purposes. The process for this can be boiled down into two main steps: identification and verification. For machine learning, this can be related to the steps of training the model and testing the model. In the training step, the model will receive data on various speakers and be able to extract specific features from the individual's voice. The system will then use these features to build a reference model that it will compare each test signal too. Once a test signal is received, the model will begin to extract the noticeable features from the signal and compare them to the database that was constructed during the training phase. Using the reference models and the features extracted from the testing data, the model can come to a decision and output an identification result based on the individual.

One unique implementation of this speaker recognition system is the use of MATLAB's object oriented programming (OOP). This allows for simple packaging of all the algorithms and signal processing steps into a single object. The authors called their class `speakerClassifier`, which creates a blueprint of a speaker classifier and simplifies the interactions that the user has to perform. Creating a new `speakerClassifier` instance based on a set of training samples is as simple as calling the constructor method and assigning the handle to a variable, and then calling the training method `train()`. The specifics of these calls are discussed in the following section.

Design of Feature Extraction

While there are some recent breakthroughs in using the raw audio samples of speakers to directly create a model of the speakers, most methods have to first extract features from the speaker samples. A common method, and the one employed in this report, is extracting the Mel Frequency Cepstrum Coefficients. The process of extracting the features is outlined in the following numbered list:

1. The audio samples are peak normalized to improve robustness
2. A pre-emphasis filter is employed to increase the energy in high frequencies

The use of a pre-emphasis filter can be ignored. However, in this report it was found that the accentuating the high frequencies improved performance and robustness slightly with negligible computational cost.

3. The full signal is zero-padded so that when it is framed, every sample is used.

If the signal were not zero-padded then when framing is done on the signal the last samples that do not fill an entire frame will not be analyzed.

4. The mean of the audio sample is centered

This step is motivated by recognizing that some portions of speech appear as random signal processes. When the random process is not centered, there will be a DC component in the spectral analysis. This DC component may make differentiating low frequency components, as well as some high frequency components if the spectral leakage is severe, from the artifacts of windowing very challenging if not impossible. While this benefit may be only for the portions of human speech that can be modeled reasonably well as a random process, the authors found that this step helped improve the overall performance.

5. The full signal is blocked into frames that may or may not overlap depending on the designer's desire.

This blocking is done by specifying a time in milliseconds for the duration of the frame, `frameDuration` and by specifying a length of time in milliseconds to slide the frame forward, `strideDuration`.

6. Each frame is windowed with a hamming window that spans the length of the frame.

This combines the purpose variable `frameDuration` as both the frame length and the window length. It gives the user the ability to balance the tradeoff between spectral frequency resolution and time resolution.

7. The discrete fourier transform (DFT) of this windowed frame is then computed with a fast-fourier transform algorithm.

The last three steps are the same taken in computing the Short Time Fourier Transform with the DFT. These three steps can be performed by using a filter bank, but the authors instead use a while loop and computations done in series. This makes for easier comprehension of the code.

8. The power spectrum is then computed by squaring the magnitude of the DFT and dividing by the length of the DFT.

It should be noted that between steps 7 and 8, the DFT is reduced to only the first half of all samples since continuing with the full DFT is computationally unnecessary.

9. The filter bank is then applied to the power spectrum to compute the Mel Frequency Spectrum (MFS) and it is scaled to deciBels

The filter bank can be tuned by specifying the variable `numFilters` when constructing the `speakerClassifier` object. The filters are evenly spaced in the Mel-Scale and therefore are non-linearly spaced in the frequency range.

10. The Discrete Cosine Transform (DCT) is performed on the MFS to obtain the Mel Frequency Cepstrum Coefficients (MFCC).

The MFCCs computed for each frame are normalized by centering the mean. This was found to improve the performance of the vector quantization algorithm.

11. The 1st and 2nd derivatives of the MFCCs are approximated by computing the differences between each frame.

This step was found to significantly improve the performance of the full speaker classification process. This is because the 1st and 2nd approximated derivatives provide direct temporal features to the vector quantization. While it is true that some aspects of time are captured in the STFT computation, the authors found that including these additional features drastically reduced misclassifications and improved robustness to noise.

Vector Quantization

The final step in speaker classification is to build models or templates of the extracted features for each speaker. If the extraction is done perfectly, each speaker should have unique features. However, due to signal noise, variations in human speech, and numerous other opportunities for errors to arrive, the feature extraction is imperfect. This means that the features can be modeled as having some randomness, suggesting that statistical analysis would be beneficial to developing these models.

Furthermore, because the designer typically chooses a very small frame duration, for example 25ms, to validate the assumption that the signal is stationary, there can be thousands to hundreds of thousands of samples in a second long audio clip. This results in a large number of features. While modern methods can now handle large sample sizes and high dimensionality, it is still helpful to reduce the number of dimensions and thereby reduce the computational load on whichever statistical model that is applied.

Both of these desires are met by the use of vector quantization. This is a method of computing a centroid that characterizes samples that appear in clusters. Vector quantization reduces the dimensions as well as the number of samples to a single set of centroids called codes. A set of codes is called a codebook. So in essence, a speaker model is just a codebook that captures the MFCCs computed for that speaker's sample.

The vector quantization algorithm that is used is the LBG algorithm. This is a computationally efficient algorithm that can accept large numbers of samples and output a codebook that has a number of codes that is a power of two. This restriction is due to the method of splitting each code into two new codes until a locally optimal fit is found and the maximum number of codes is reached. The designer can specify the maximum number of codes used in this algorithm by defining the `numClusters` variable when constructing an instance of the `speakerClassifier` class.

Performance and Robustness

For our first test of the system, we established a benchmark of human performance of the system in order to compare the performance to the machine learning model. In order to get this data, both of us "trained the model" by listening to each of the eight test sound files. Then we executed the test by listening to each sound file individually and trying our best to match it to the corresponding file number based on what we have "stored" in our memory. For our results, both of us had a 75% accuracy for being able to guess the speaker based on the sound file given to us.

Our second test related to being able to access the original signal correctly and displaying it in the time domain. Our output for a single audio file looks like this:

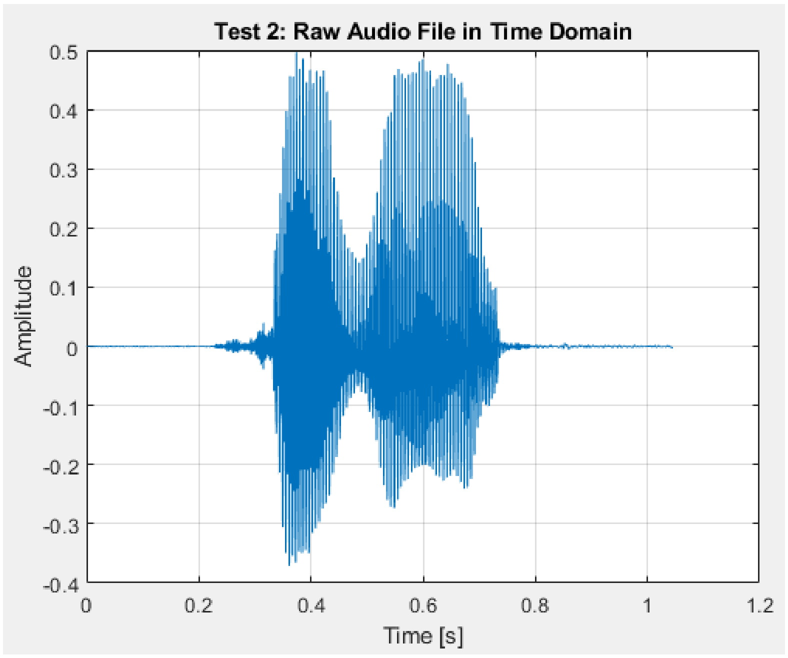


Figure 1: Time Domain Representation of the First Training Speaker

As is visible here, the raw data is over a relatively long period of time, and needs to be normalized. Based on our code, we were able to extract a sampling frequency of 12500 Hz, and with that found that a frame of 256 samples takes 20.48ms.

For our third test, we wanted to output the filter banks that were created using our mel-spaced frequency function. As mentioned earlier, the filter banks are linearly spaced for lower frequencies and are logarithmically spaced for higher frequencies. The graph is shown here :

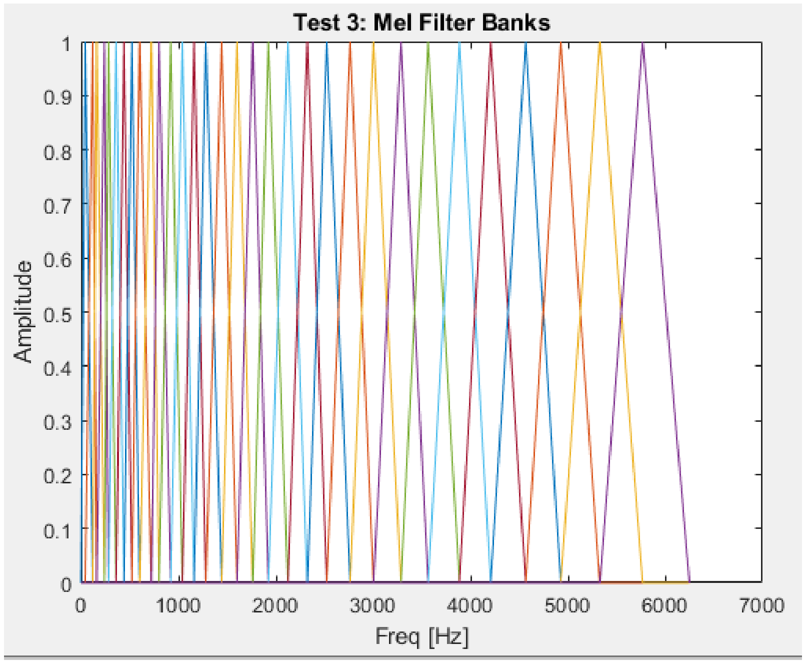


Figure 2: Linear Frequency Representation of Mel Filter Banks

Our fourth test involved computing and plotting the speech spectrum before and after the mel-frequency wrapping. The two spectrograms are shown below:

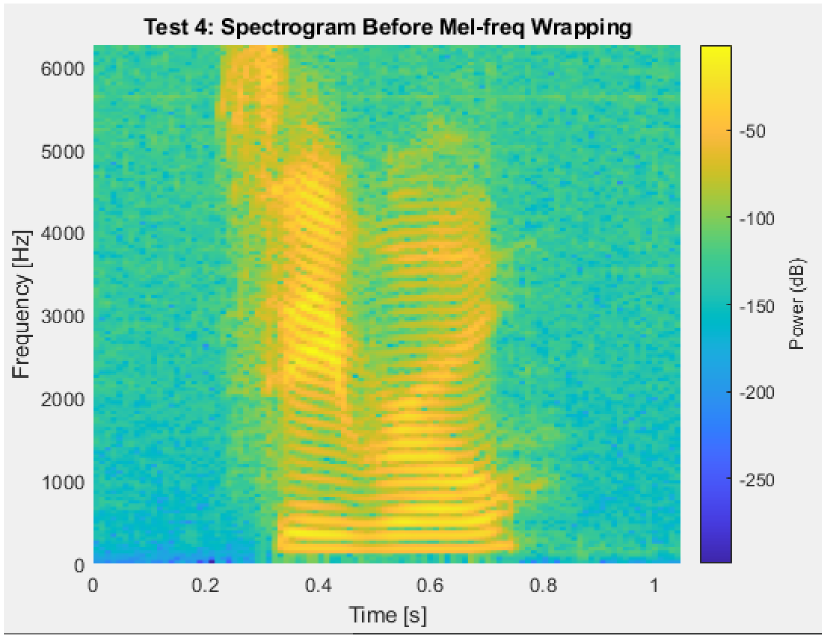


Figure 3: Spectrogram of the First Training Speaker Sample

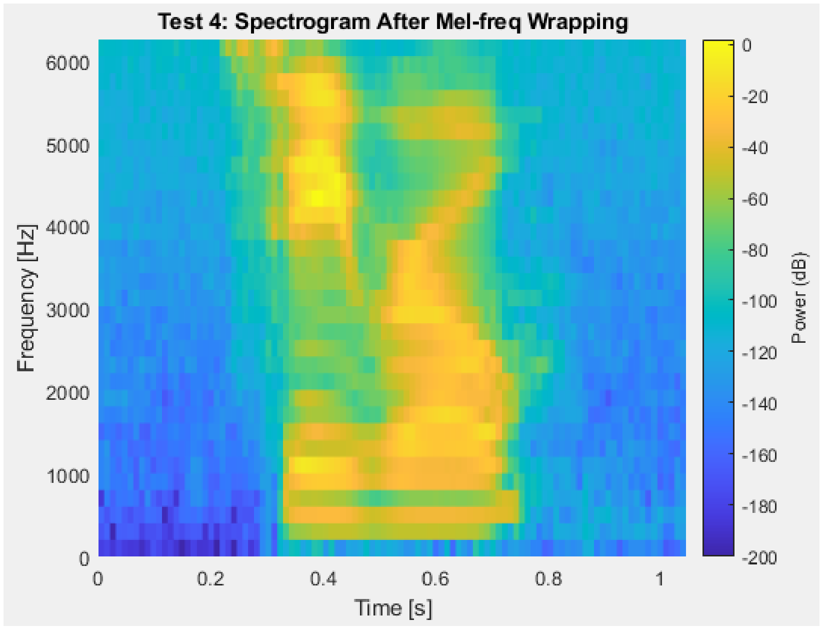


Figure 4: Spectrogram of the First Training Speaker Sample after Applying Mel Frequency Banks

Our fifth test revolved around outputting the cepstrogram, or the spectrogram of the MFCC's, as well as plotting the MFCC vectors of two different speakers to show the differences in their speech patterns:

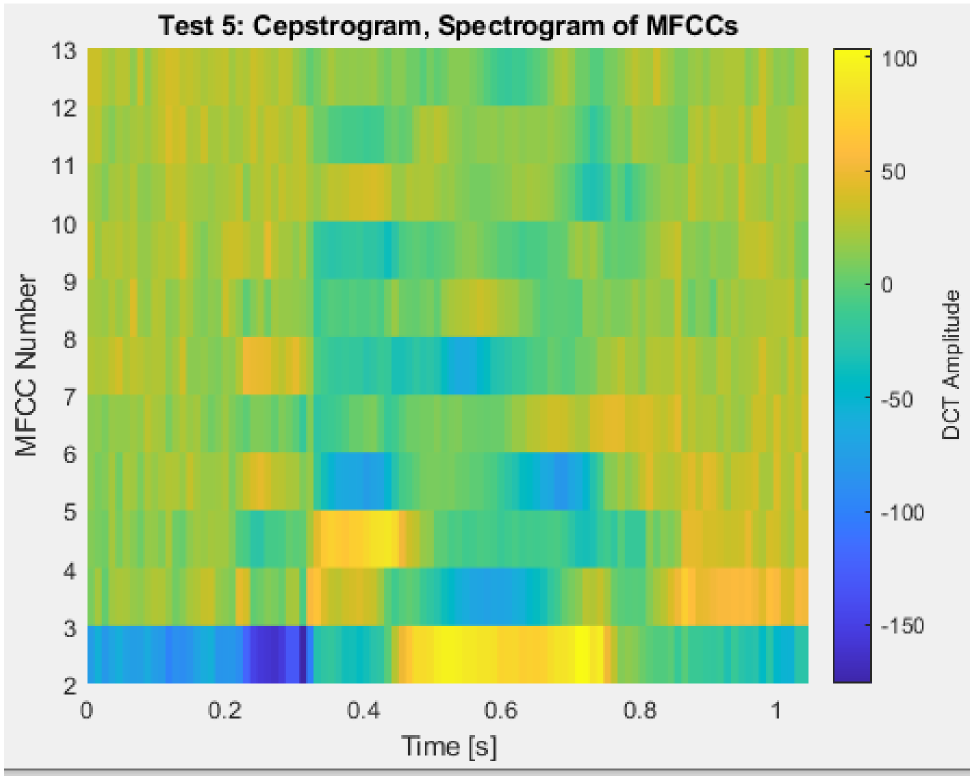


Figure 5: Spectrogram of the First Training Speaker Sample after Extracting MFCCs

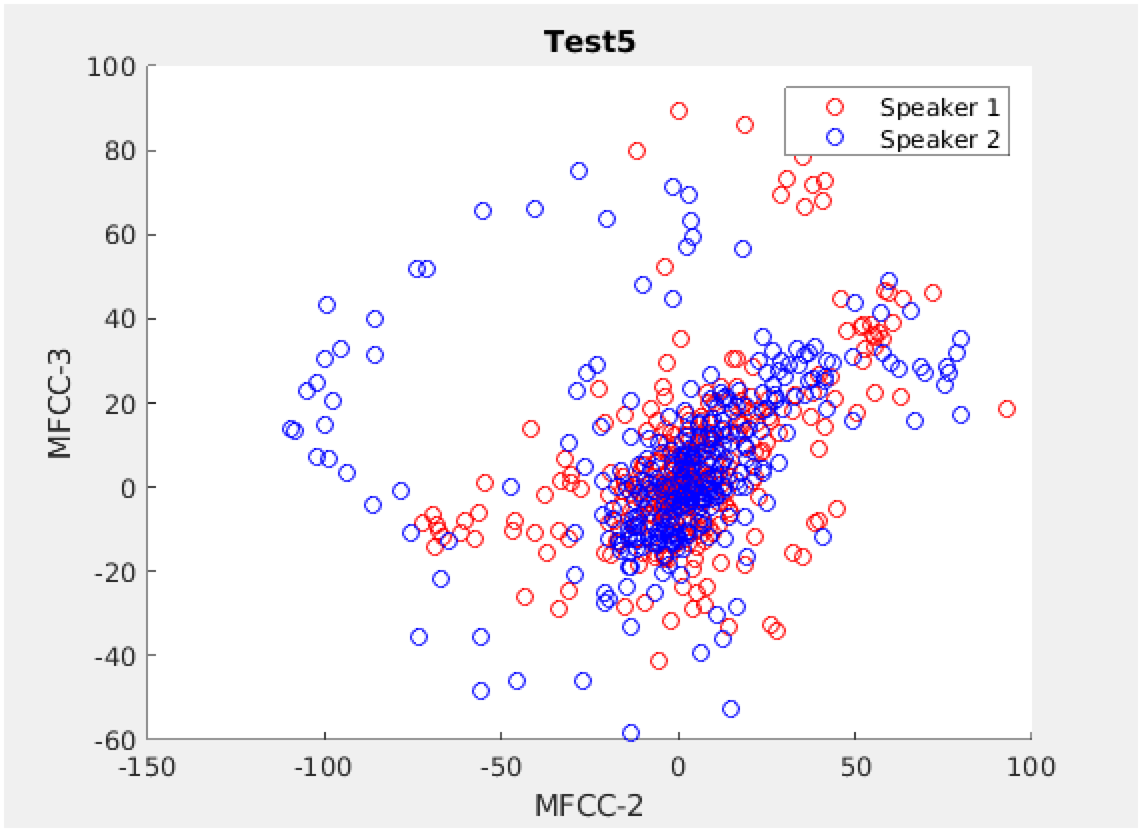


Figure 6: 2D Scatter Plot of MFC Coefficients 2 and 3 for Training Speaker 1 and 2

As can be seen in the graph directly above, the red and blue points mark the MFCC's of two different speakers. Even though the majority of both speakers can be seen to have similar data in a small area, there are other differences that can be seen through the outlying points on the graph. These differences are something we used in order to match the test data to the data that was collected in our model.

Our next test involved showing the output of our LBG algorithm that categorized the MFCCs into centroids. We wanted to show the output of those centroids to show what our model will use to match the different components of the different speech signals. The graph is shown below:

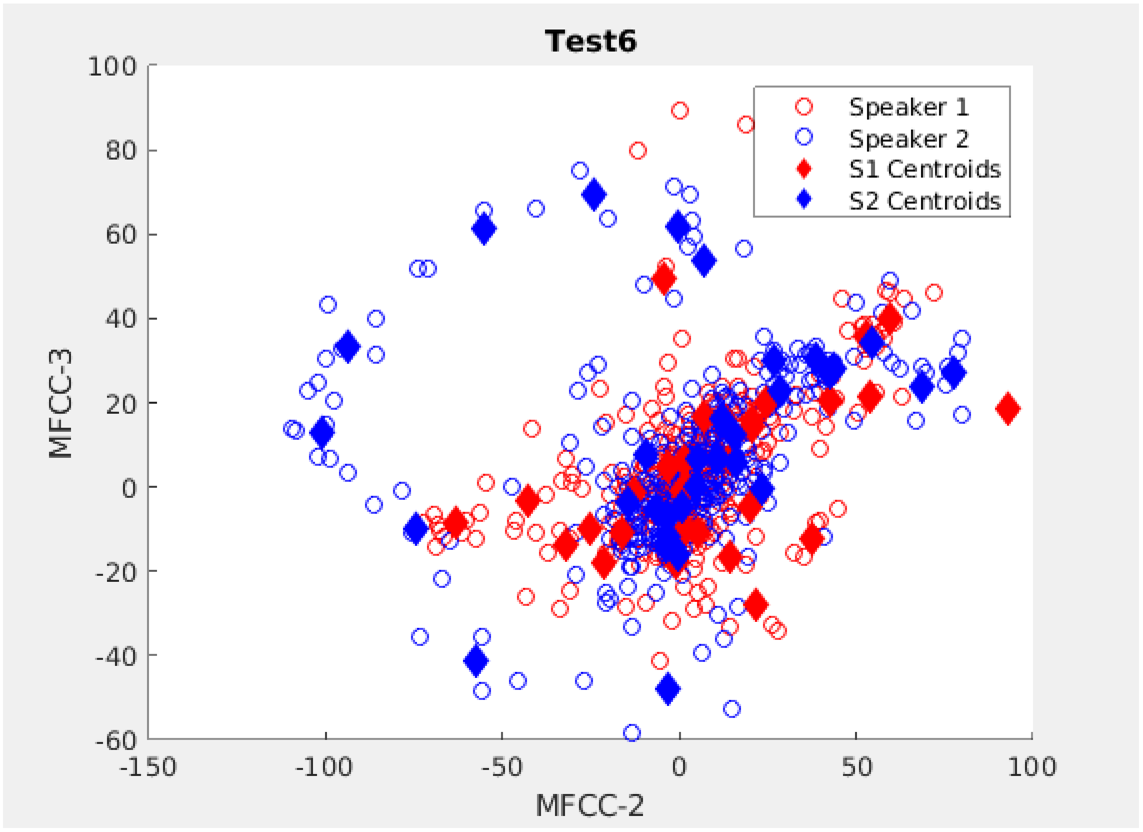
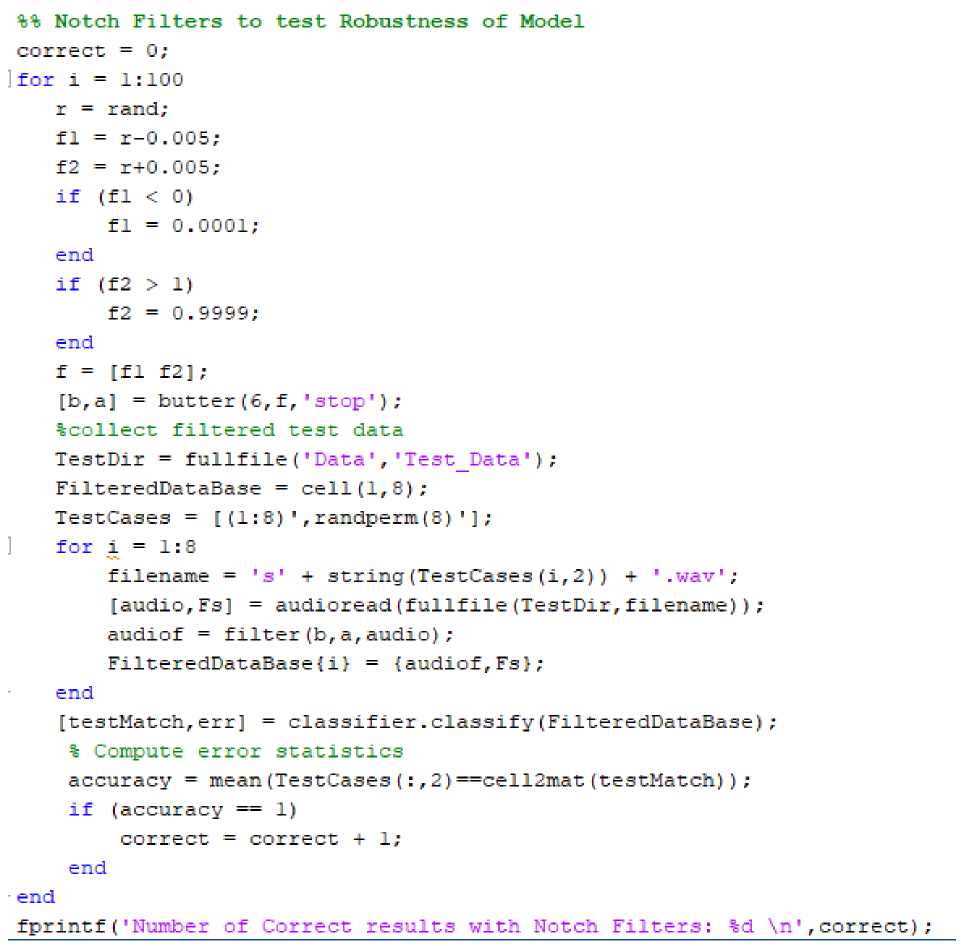


Figure 7: Scatter Plot of MFCCs 2 & 3 Clustered around Training Speaker 1 and 2 Centroids

As shown above, the LBG algorithm grouped the MFCCs of both speakers and marked the centroids created using the corresponding color diamond. As mentioned before, even though there are both red and blue diamonds in one major area, the outliers will also help us distinguish between the speakers.

Our next test recorded the results of the system in terms of its recognition rate. To accomplish this we randomized the order of the eight test files. After the model had been trained, we passed the random order speech files and had the program output the speaker numbers it thought it was based on the sound file. Using the eight given test files and multiple iterations of the code and randomization, our system had an accuracy of 100% each time the file was run. This means our model was successfully able to recognize key features in the speaker's speech pattern and match those features to data that was stored during the training process.

Once we had the results from the given files, we wanted to test the robustness of our system by using notching filters, bandstop filters with narrow stop bands, to eliminate different frequency components from the original speech file and testing the accuracy. To do this, we randomly selected a frequency and created a stopband by adding and subtracting a constant to that frequency. From there, we created a notching filter and filtered the test data before passing it to the model that had been trained with the "clean" data that was not filtered and recording the accuracy. We ran this test 100 times and recorded every time the system was able to get 100% accuracy for the filtered signals.



```
%% Notch Filters to test Robustness of Model
correct = 0;
for i = 1:100
   r = rand;
   f1 = r-0.005;
   f2 = r+0.005;
   if (f1 < 0)
       f1 = 0.0001;
   end
   if (f2 > 1)
       f2 = 0.9999;
   end
   f = [f1 f2];
   [b,a] = butter(6,f,'stop');
   %collect filtered test data
   TestDir = fullfile('Data','Test_Data');
   FilteredDataBase = cell(1,8);
   TestCases = [(1:8)',randperm(8)'];
   for i = 1:8
       filename = 's' + string(TestCases(i,2)) + '.wav';
       [audio,Fs] = audioread(fullfile(TestDir,filename));
       audiof = filter(b,a,audio);
       FilteredDataBase{i} = {audiof,Fs};
   end
   [testMatch,err] = classifier.classify(FilteredDataBase);
    % Compute error statistics
    accuracy = mean(TestCases(:,2)==cell2mat(testMatch));
    if (accuracy == 1)
        correct = correct + 1;
    end
end
fprintf('Number of Correct results with Notch Filters: %d \n',correct);
```

The code above is for a notch filter that has a stopband that is plus and minus 0.005 omega from the randomly generated frequency. Based on this test, the model was able to get 100% accuracy about 53-60% of the time. This range will change with each iteration since the frequencies selected are random, but will be around this range. Changing the width of the stop band will also change the results, but will be random as well since the original frequency is random. It is obvious from these results that by eliminating certain frequencies from the speech signal will remove key points of data that the model uses to match the speaker to the data stored from the training, causing the model to fail.

Conclusions

Based on our model, we were able to successfully create a system that could recognize the speech patterns of the eleven given individuals, and extract significant features from the data. From there, when presented with randomly selected speech files from the training data, the model was able to extract the features and match them to the data already collected on all the speakers. With the given data set, our model was able to get 100% accuracy on every iteration of the code.

Furthermore, we wanted to test the robustness of our model when frequencies were filtered out from the testing data to see if it would affect the accuracy of the results. In fact we found that there are frequencies that are essential to our model in being able to successfully match the test data to reference models already created. This was seen by the model failing when running the filtered test data through the model.