Kaleb Kokott and Hana Shaik
Professor Ding
EEC 201
March 15, 2025

EEC 201 Final Project Report - Kokott and Shaik

I. Approaches

A. Speech Processing

For the speech processing component of the project, we needed to implement a pipeline consisting of five blocks: frame blocking, windowing, FFT, mel-spectrum, and cepstrum.

Frame blocking consisted of separating the data into chunks of a fixed length (denoted "N"), which we opted to pass into the function as a parameter. For blocks at the end of the data, where the signal may not have had enough data points to fully fill a block of length N, the resulting block was zero-padded to the proper length. Furthermore, if any blocks were identically zero throughout, these blocks were removed from the output set, as these empty blocks would mess up the measurement.

For the windowing, we applied a Hamming window (equation $w(n)=0.54-0.46*\cos(2\pi n/(N-1))$, in order to make the data "smoother" at the ends and prevent artifacts from showing up, then element-wise multiplied the window with the frame-blocked data chunk.

For the FFT block, our approach was simply to compute the N-point FFT of each of the windowed frames from the previous block using MATLAB's FFT function.

For the mel-spectrum block, we first used the provided melfb.m function to obtain the matrix for the mel-spaced filterbank. We then obtained the indices for the positive-frequency coefficients of the FFT by dividing N by 2 and taking the floor of that value and adding 1 to it. We then generated an N-point array from 0 to half of the sampling frequency for plotting the mel-spaced filterbank responses and the power spectrum before mel-wrapping. If the showFigures parameter was set to True, we then wrote code to generate plots of the mel-spaced filterbank responses and the power spectrums before mel-wrapping. We then finally obtained the mel spectrum by multiplying the mel-spaced filterbank matrix with the absolute value of the FFT output squared. We then plotted this result as the power spectrum after mel-wrapping if showFigures was set to True.

For the cepstrum block, we were given this equation to compute the cepstrum:

$$\tilde{C}_n = \sum_{k=1}^{K} (\log \tilde{S}_k) \cos \left[\frac{n\left(k - \frac{1}{2}\right)\pi}{K}\right], n = 0, 1, \ldots, K-1$$

,

where $\tilde{S}_k$ represents the mel spectrum values and the equation itself represents the DCT of the mel-spectrum. We thus first took the log of the mel spectrum results to match the equation provided. We then originally used MATLAB's DCT function to calculate the DCT of the previous result. However, we found that MATLAB's DCT function did not quite match the DCT equation provided above. We thus replaced our use of the DCT function with this equation exactly as shown to calculate the DCT of this result. We then ignored the first coefficient of this result to account for volume changes and returned the cepstrum, which is the final desired result of the speech processing pipeline.

The MFCC function is the overall function that takes an audio file as an input and outputs the Mel Frequency Cepstrum Coefficients. It starts by reading the audio file, then passes that data into the frame-blocking function, taking the output of the frame-blocking and passing it into the windowing function, then taking the output of the windowing function and passing it into the processing function, at which point it optionally displays an image of the MFCC (color-intensity map) and outputs the set of vectors describing the MFCCs.

B. Vector Quantization

For the LBGTraining function, we start by finding the mean of all the vectors included, then proceed to split the mean, effectively dividing the data into two sets based on their proximities to the mean. Next, the new mean of each set is calculated, and the sets are recalculated according to the new means. This process of calculating the means and adjusting the sets is repeated until the change in sets is sufficiently small (as defined by parameter "E"). Finally, the process of splitting and adjusting means is repeated until we have the desired number 2^Iterations sets and means, and we call each mean a "codeword", part of the specific audio file's "codebook".

ProjectTraining1 sets important parameters such as N=512 (size of frame block), M=200 (size of step between starting points for the frame block), K=20 (size of MFCC vector), and NumIterations=3 (2^3=8 codewords per audio file). It also automatically collects all .wav files of the form "s*.wav", where "*" represents any string such as "1", "12", or "tore". However, note that the file directory specified only has files of the form s1.wav, s2.wav, … s11.wav. Therefore, we can ensure that the answer output matches the speaker ID we intuitively would expect it to by reading the number from the file name, and assigning the MFCC

coefficients and codewords to that index in their respective arrays. Otherwise, we may end up with situations where “s11.wav” is in index 2, so the function would output “2” instead of “11”.
The ProjectTraining1 function runs each audio file through the MFCC function, then through the LBGTraining function, storing the codewords output for each audio file separately. The next step is to compute the MFCCs for the test data, performed in a similar manner to the training data, and compare the distances from the MFCCs to each set of codewords to produce an estimate as to who was speaking.

For ProjectTraining2, we first needed to make a slight modification to the files provided in the Twelve-Training and Twelve-Testing folders. We found that in these folders, there was no speaker 5, i.e., the numerical labeling of the speakers in these folders was 1-4 and 6-19 as opposed to 1-18. This was generating an error because of the way we performed indexing in ProjectTraining1. The code was unable to run to completion at this point. We felt that the simplest method for resolving this error was renaming speakers 6-19 to 5-18, i.e., reducing each of the speaker ID numbers for speakers 6-19 by 1.

We then made a few modifications to our ProjectTraining1 code to obtain our code for ProjectTraining2. First, we changed the file paths in lines 5 and 33 to "Twelve-Training\Twelve_train*.wav" and "Twelve-Testing\Twelve_test*.wav," respectively. We then changed lines 24 and 38 such that the content to remove from the filenames was [".wav","Twelve_train"] and [".wav","Twelve_test"] as the files in these directories were named differently from the files in the original Training_Data and Test_Data directories.

To improve the accuracy of the speech recognition code for this dataset, we modified a few of our parameters. We increased K from 20 to 35 and increased NumIterations from 3 to 4. Other than the modifications described above, our code in ProjectTraining2 is identical to our code in ProjectTraining1.

For ProjectTraining3, we first needed to make a slight modification to the files provided in the Zero-Training and Zero-Testing folders. We found that in these folders, there was no speaker 5, i.e., the numerical labeling of the speakers in these folders was 1-4 and 6-19 as opposed to 1-18. This was generating an error because of the way we performed indexing in ProjectTraining1. The code was unable to run to completion at this point. We felt that the simplest method for resolving this error was renaming speakers 6-19 to 5-18, i.e., reducing each of the speaker ID numbers for speakers 6-19 by 1.

We then made a few modifications to our ProjectTraining1 code to obtain our code for ProjectTraining3. First, we changed the file paths in lines 5 and 33 to "Zero-Training\Zero_train*.wav" and "Zero-Testing\Zero_test*.wav," respectively. We then changed lines 24 and 38 such that the content to remove from the filenames was [".wav","Zero_train"] and [".wav","Zero_test"] as the files in these directories were named differently from the files in the original Training_Data and Test_Data directories. Other than the modifications described above, our code in ProjectTraining3 is identical to our code in ProjectTraining1.

For ProjectTraining4, we made a few modifications to our ProjectTraining1 code to obtain our code for ProjectTraining4. First, we changed the file paths in lines 5 and 33 to "Eleven Training\s*.wav" and "Eleven Test\s*.wav," respectively. Other than the modifications described above, our code in ProjectTraining4 is identical to our code in ProjectTraining1.

For ProjectTraining5, we made a few modifications to our ProjectTraining1 code to obtain our code for ProjectTraining5. First, we changed the file paths in lines 5 and 33 to "Five Training\s*.wav" and "Five Test\s*.wav," respectively. To improve the accuracy of the speech recognition code for this dataset, we increased K from 20 to 30. Other than the modifications described above, our code in ProjectTraining5 is identical to our code in ProjectTraining1.

For ProjectTraining6, we first defined a function called notchFilterData.m. This function was written in order to take the data from Test_Data, read that data, put that data through a notch filter, and then create new audio files containing that notch-filtered data.

ProjectTraining6 is identical to ProjectTraining1, except that it calls this notchFilterData function to generate the notch-filtered test data. In ProjectTraining6, we also changed the I2=dir line to use the Filtered_Test_Data directory instead of the Test_Data directory so that the code is tested on the notch-filtered test data instead of the original test data.

The purpose of ProjectTraining7.m was to not only test our code's ability to identify speakers, but also to test its ability to identify the word that is said in the audio files. This code is trained on both Twelve-Training and Zero-Training and tested on both Twelve-Testing and Zero-Testing. Originally, since the best K value and best NumIterations value for each of these sets was different, we tried to set different values for K and NumIterations, using K_twelve=35 and

NumIterations_twelve=4 for the twelve dataset and K_zero=20 and NumIterations=3 for the zero dataset. However, because we used disteu.m to determine whether the spoken word was “twelve” or “zero” and disteu.m requires consistency in the sizes of the inputs, we eventually had to set the K values to the same number and NumIterations to the same number. We chose K=35 and NumIterations=4 in order to maximize the accuracy in identifying the speakers in the twelve dataset, which sacrificed a little bit of the accuracy in identifying the speakers in the zero dataset.

In order to determine the spoken word, we added some logic that would essentially calculate the distance and distortion between the data and the codewords for both the twelve dataset and the zero dataset. Based on those distortion calculations, we predicted whether the spoken word was “twelve” or “zero.” We added some code at the end to both print the ID of each speaker as well as the spoken word for each speaker.

In ProjectTrainingTests2.m, we tried out an alternative approach for training and testing the code on the combined twelve and zero datasets. We made our model more robust to the names of the audio files, basically allowing us to run the code directly on the original twelve and zero datasets without having to rename the files as we did for ProjectTraining2.m, ProjectTraining3.m, and ProjectTraining7.m. This code also provided a different approach for running the code on the testing data and determining the accuracy of the model compared to how we did this in ProjectTraining7.m.

II. Tests

We performed seven tests to verify the accuracy of our project code. The first test was performed on the default speaker set. We trained our model on the audio recordings of the 11 default speakers saying “zero” from the Training_Data folder and tested the model on the audio recordings of the 8 default speakers saying “zero” from the Test_Data folder. We also performed some additional subtests here. The default datasets were augmented with training and testing data from Kaleb and his sister and the code was tested in that case. The default datasets were also later augmented with 10 random speakers from the Zero-Training and Zero-Testing datasets to test its accuracy in that case.

For our second test, we trained our model on the audio recordings of the 18 students saying “twelve” from the “Twelve-Training” folder and tested the model on the audio recordings of the 18 students saying “twelve” from the “Twelve-Testing” folder.

For our third test, we trained our model on the audio recordings of the 18 students saying "zero" from the "Zero-Training" folder and tested the model on the audio recordings of the 18 students saying "zero" from the "Zero-Testing" folder.

For our fourth test, we trained our model on the audio recordings of the 23 students saying "eleven" from the "Eleven Training" folder and tested the model on the audio recordings of the 23 students saying "eleven" from the "Eleven Test" folder.

For our fifth test, we trained our model on the audio recordings of the 23 students saying "five" from the "Five Training" folder and tested the model on the audio recordings of the 23 students saying "five" from the "Five Test" folder.

For our sixth test, we trained our model on the audio recordings of the 11 default speakers saying "zero" from the Training_Data folder and tested the model on the audio files of the 8 default speakers saying "zero" after those files were passed through a notch filter from the Filtered_Test_Data folder.

For our seventh test, we trained our model on the audio recordings of the 18 students saying "twelve" from the "Twelve-Training" folder as well as the audio recordings of those 18 students saying "zero" from the "Zero-Training" folder. We tested the model on the audio recordings of the 18 students saying "twelve" from the "Twelve-Testing" folder and tested the audio recordings of the 18 students saying "zero" from the "Zero-Testing" folder.

III. Results

For our first test, we obtained 100% accuracy. The test speakers were fed to the model in numerical order from 1 through 8 and at the very end of the code in ProjectTraining1.m, we printed the speakerID list, indicating what number the model identified each speaker as. The speakerID list was printed as "1 2 3 4 5 6 7 8," indicating that the model correctly identified each speaker. Thus, for the first test, we obtained 100% accuracy as desired. For both subtests, we also attained 100% accuracy in speaker recognition.

For our second test, we also obtained 100% accuracy. The test speakers were fed to the model in numerical order from 1 through 18 and at the very end of the code in ProjectTraining2.m, we printed the speakerID list, indicating what number the model identified each speaker as. The speakerID list was printed as "1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18," indicating that the model correctly identified each speaker. Thus, for the second test, we obtained 100% accuracy as desired.

For our third test, we were unable to obtain 100% accuracy. The test speakers were fed to the model in numerical order from 1 through 18 and at the very end of the code in ProjectTraining3.m, we printed the speakerID list, indicating what number the model identified each speaker as. The speakerID list was printed as "1 2 3 4 5 10 7 8 9 10 11 12 13 14 15 16 17 18," indicating that the model correctly identified each speaker, except for speaker number 6, who was erroneously identified as speaker number 10. We tried modifying parameters such as K for the number of MFCC bins, NumIterations for the number of codewords to generate in the LBG algorithm, and M for the distance between frame starts. However, none of these modifications changed the results and speaker number 6 was always being erroneously identified as speaker number 10. Thus, for the third test, the code was successfully able to identify 17 out of 18 speakers and we obtained about 94% accuracy.

For our fourth test, we obtained 100% accuracy. The test speakers were fed to the model in numerical order from 1 through 23 and at the very end of the code in ProjectTraining4.m, we printed the speakerID list, indicating what number the model identified each speaker as. The speakerID list was printed as "1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23," indicating that the model correctly identified each speaker. Thus, for the fourth test, we obtained 100% accuracy as desired.

For our fifth test, we also obtained 100% accuracy. The test speakers were fed to the model in numerical order from 1 through 23 and at the very end of the code in ProjectTraining5.m, we printed the speakerID list, indicating what number the model identified each speaker as. The speakerID list was printed as "1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23," indicating that the model correctly identified each speaker. Thus, for the fifth test, we obtained 100% accuracy as desired.

For our sixth test, we obtained 100% accuracy. The test speakers were fed to the model in numerical order from 1 through 8 and at the very end of the code in ProjectTraining6.m, we printed the speakerID list, indicating what number the model identified each speaker as. The speakerID list was printed as "1 2 3 4 5 6 7 8," indicating that the model correctly identified each speaker. Thus, for the first test, we obtained 100% accuracy as desired.

For our seventh test, we obtained 100% accuracy in identifying the spoken word and 94% accuracy in identifying the speaker. The code correctly identified the spoken word in the first 18 testing audio files as "twelve" and the spoken word in the second 18 testing audio files as "zero." The first 18 speakers were correctly identified 1 through 18 from the twelve dataset. However, from the zero dataset, speaker 6 was incorrectly identified as speaker 16 and speaker 11 was incorrectly identified as speaker 12. Speaker 6 is persistently misidentified but speaker 11 was misidentified because of the inability to use different K and NumIterations values for each set so that would be something to potentially look into in the future, the ability to use different values for each set.

Nonetheless, our model was highly accurate across all tests.

IV. Unique Efforts/Contributions

For the project code, Hana wrote the code for the FFT, mel-spectrum, and cepstrum blocks of the speech processing pipeline. Hana's code is entirely encapsulated within the processingpart2.m file in this repository.

Kaleb wrote the code for the frame blocking and windowing blocks of the speech processing pipeline. This code is encapsulated in the FrameBlocking.m and Window.m files in the repository. Kaleb also wrote the code in MFCC.m to put all of the speech processing components together. Kaleb also wrote the code for LBGTraining.m and ProjectTraining1.m. Kaleb also performed the testing and debugging of our code on the original, default dataset.

Hana wrote the code for ProjectTraining2.m, ProjectTraining3.m, ProjectTraining4.m, and ProjectTraining5.m based on Kaleb's code for ProjectTraining1.m. Hana thus also performed the testing and debugging of our code on the four other datasets described above.

Hana wrote the code for notchFilterData.m as well as for ProjectTraining6.m. Hana thus also performed the testing and debugging of our code for the notch-filtered test data. Hana also wrote the code for ProjectTraining7.m and thus tested and debugged the code for that case. Kaleb wrote the code for ProjectTrainingTests2.m and thus also tested and debugged the code for that case.

Hana also added comments to the disteu.m and melfb.m files and also wrote the last section of this report. Kaleb and Hana wrote the first three sections of this report together.