# Abstract

This paper presents a comprehensive study of a voice recognition system developed using Mel Frequency Cepstrum Coefficients (MFCC). The system design and methodology are explored in detail, with a focus on feature extraction and the use of MFCC. Performance is evaluated under various conditions, providing insights into its robustness and accuracy. While the system performs exceptionally well on homogeneous datasets, achieving 100% accuracy, performance drops slightly on non-homogeneous datasets. For signal loss within certain frequency bands, the system retains reasonably well. In conclusion, the speaker recognition model has great performance with the small dataset provided and has areas for future improvements.

# Introduction

In the era of digital transformation, voice recognition has emerged as a pivotal technology, driving advancements in various sectors such as healthcare, automation, and security. This paper presents a comprehensive study of a voice recognition system developed using Mel Frequency Cepstrum Coefficients (MFCCs) and the LBG algorithm, a well-established method in the field of speech and audio processing \cite{lgb\_paper}.

A detailed overview of the speaker classification processes used in this paper is provided as well as an explanation of the software environment. Selection of the hyperparameters is vital to the performance of the system, and an evaluation of different parameters is demonstrated. After a thorough testing of the system under various conditions was completed, a clear view of the strengths and weaknesses was summarized. With these insights, additional paths for further improvements are discussed.

# Methodology

The central principle of voice recognition, and indeed of all classification, is feature extraction: determining or deciding which properties of the data are most discriminatory, i.e. contain the most information about the class of the data. For voice recognition, an initial approach might be to use the spectrogram of the sound; however, the features of the spectrogram are highly dependent on the specific words being spoken, the speed of articulation, and information about the timbre of the voice, while present, is still buried.

A more successful method is to use MFCCs. The training data is divided into frames of a given length with a given amount of overlap. These frames are windowed to mitigate the spectral distortion which would arise from abrupt changes at the beginning and end of the frame. Fourier Transform of each frame is taken, and the strength of the signal in several frequency bins is found. The exact number of bins used are critical in the efficacy of the overall recognition scheme. A popular principle is to use the Mel-frequency scale, which is designed to reflect how human aural acuity changes with frequency: it uses linear spacing below 1kHz and logarithmic spacing above.

Once the Mel-spectrum values are obtained for each frame, the logarithmic values are converted from one frequency domain to another frequency domain via the Discrete Cosine Transform (DCT). The result of transforming a frequency spectrum twice is known as cepstrum, and for this project, the cepstrum coefficients facilitate disentangling the vocal characteristics from the essential sound sequence.

After the MFCCs are found for each frame, these sampled values are best thought of as vectors in a n-dimensional space, where n is the number of mel-scaled bins used. A person’s voice will tend to produce vectors of a certain type, or types, corresponding to the qualities of their voice. These samples can then be plotted and analyzed via statistical methods. For this paper, the LBG clustering algorithm is used to group samples and compare group distances to other audio files \cite{lgb\_paper}.

A centroid is defined as the mean of a grouping of MFCC samples. When the algorithm begins, it will start with a mean for the entire sample space, and then be split into two centroids. The new centroids will be placed equal distance apart in opposite directions. Samples will then be grouped to the nearest centroid, and the centroids will move toward the mean of the groupings. Centroids will continuously regroup with samples and move toward the mean until the movement is below a threshold. Then the two centroids are split, and the entire process repeats until a certain number of centroids is attained. When the algorithm finishes, the final centroids are saved as a codebook for that audio file.

A codebook is generated for each audio file in the training set. To classify any given audio file, it is split into frames and MFCCs are found for each frame. These MFCC vectors are then compared to each codebook by finding the distance from each sample to the nearest centroid. Whichever codebook has centroids with the smallest overall distance to the test MFCCs, this will be considered as the predicted label. The speaker whose audio file was used to train the predicted codebook label will be the predicted speaker.

# Design Choices

For the software design, a codebook class was created to hold the centroid locations. This class will handle the method of feature extraction, clustering algorithm, and saving of the centroid locations in memory. With the codebook class facilitating training of speaker models, a new set of data can then be compared to each of the codebook classes to get distance from centroids.

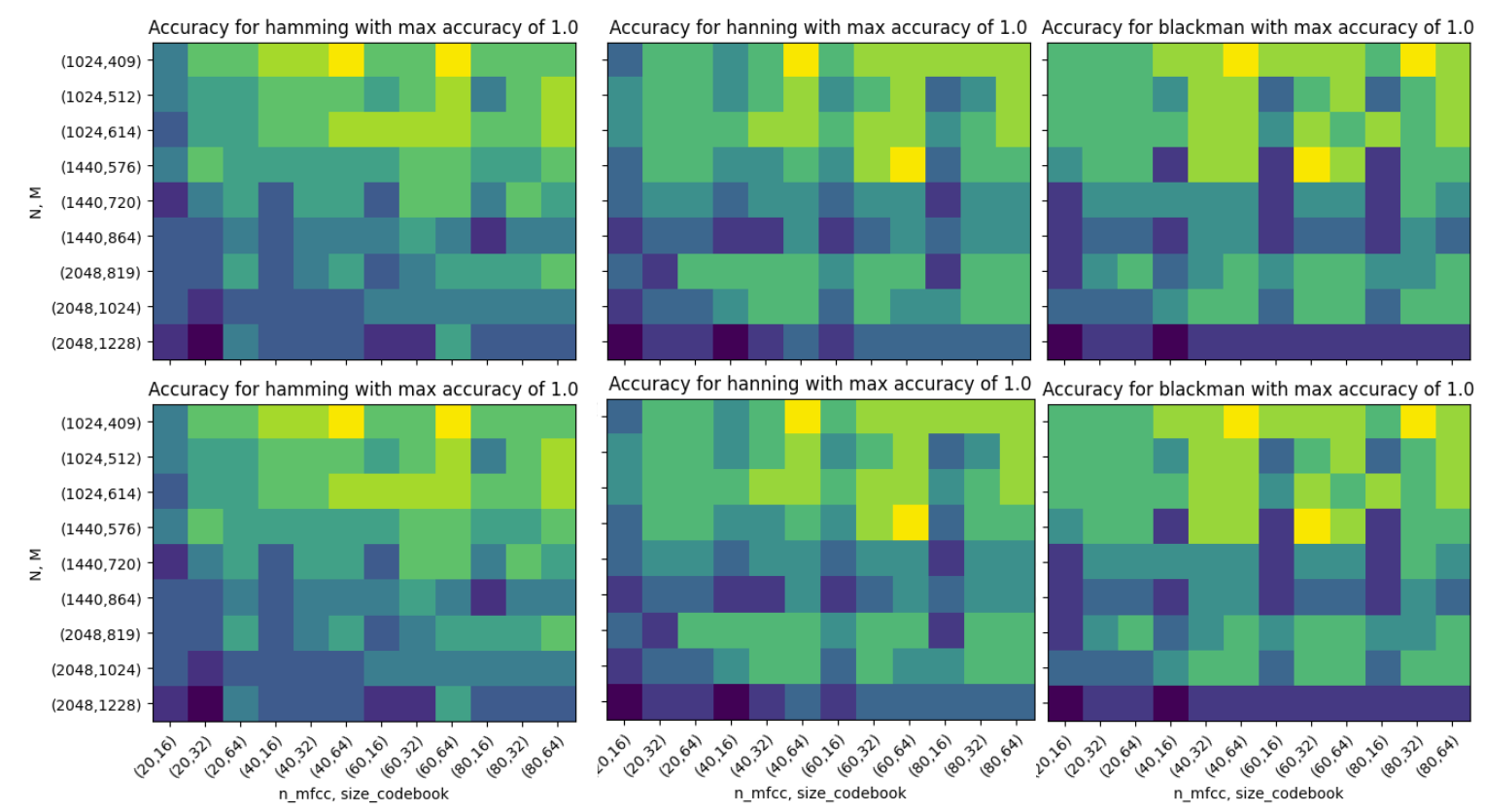
The class “codelibrary” behaves as an organizer of the codebooks; it has methods for creating and saving a set of codebooks corresponding to a set of training data in a given folder. Its prediction method finds the codebook which has the smallest distance from the given data. There is even a get accuracy method to run predictions on an entire folder of audio files and take the average prediction accuracy.

## Hyperparameter Sweep

The core system described in the Methodology section has several hyperparameters which can be tuned to improve performance on a specific instance of the speech recognition problem. These hyperparameters, and their variable representations in our code, are described in the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Hyperparameter** | **Meaning** | **Variable** | **Hyperparameter Range** |
| Frame Length | Number of Samples in Frame | N | 1024, 1440, 2048 |
| Hop Size | Number of Samples between start points of subsequent frames | M | 0.4, 0.5, 0.6 (fraction of N) |
| Window | Type of window applied to frame | window | Hamming, Hanning, Blackman, Bartlett |
| Number of MFCCs | Corresponds to the number of frequency bins | n\_mfcc | 20, 40, 60, 80 |
| Codebook Size | Number of centroids | size\_codebook | 16, 32, 64 |

To improve the performance of our system on the in-class recordings, we performed a hyperparameter sweep over the values given in the Hyperparameter Range column of the table. Figure \ref{sweep\_param\_fig} contains an example of the results.



For each window, the hyperparameters $N=1024$, $M=409$, $n\\_mfcc$ = 40, and $size\\_codebook = 64$ resulted in perfect accuracy, so those parameters were selected for the system. These hyperparameters were found for audio data recorded with a sampling rate of 48kHz, and alternative sampling rates would need to have updated $N$ and $M$ values.

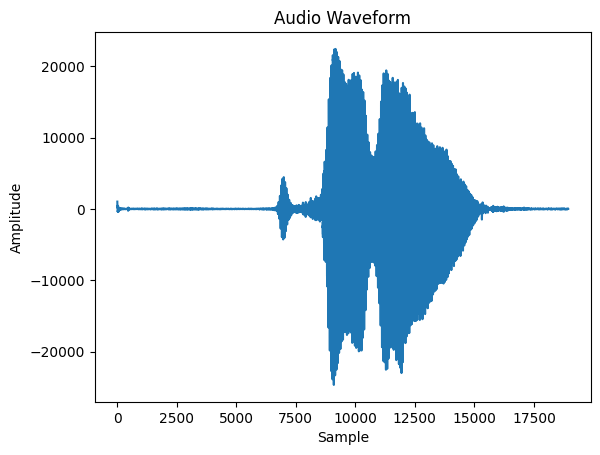
# Tests

## Test 1

Randall played the entire training dataset for Conor to familiarize himself with, and then played the test dataset. Conor correctly identified 3 of the 8 test audio files, giving us a baseline human accuracy of 0.375.

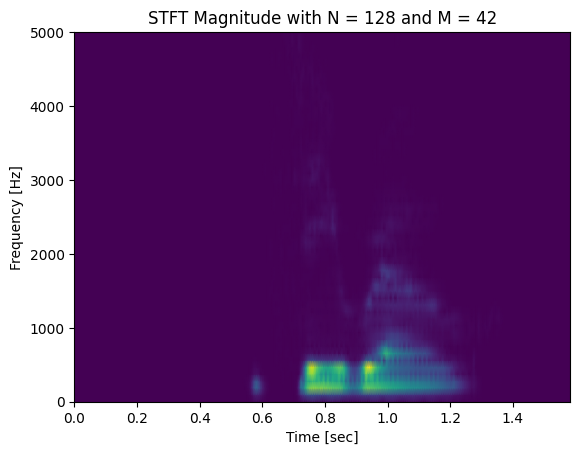
## Test 2

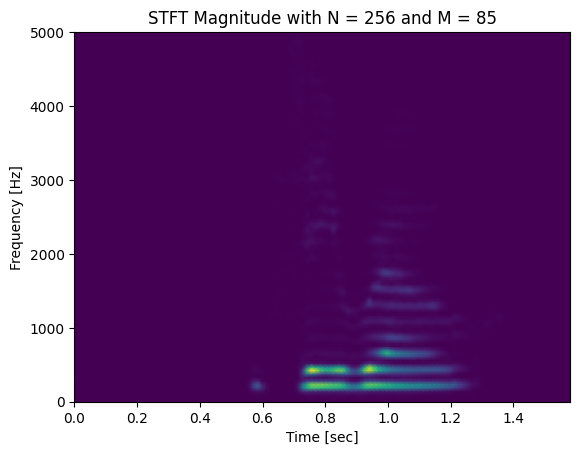
Our Test.ipynb file makes it easy to play and plot any of the audio files in the provided data. For example, here is a plot of the 2nd of the audio files in the test set.

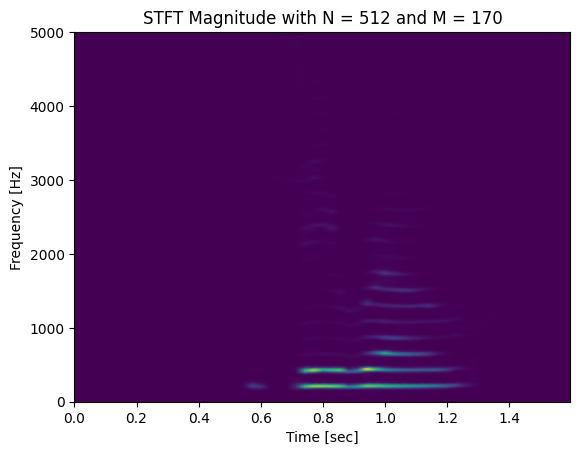


The sampling rate of the given data is 12500; at this sampling rate, 256 samples corresponds to 20.5 ms.

The last of the 8 test audio files could not be read originally; we exported and resaved it in a different program. This fixed the issue, but it did result in a sampling rate of 12000 for that file specifically.



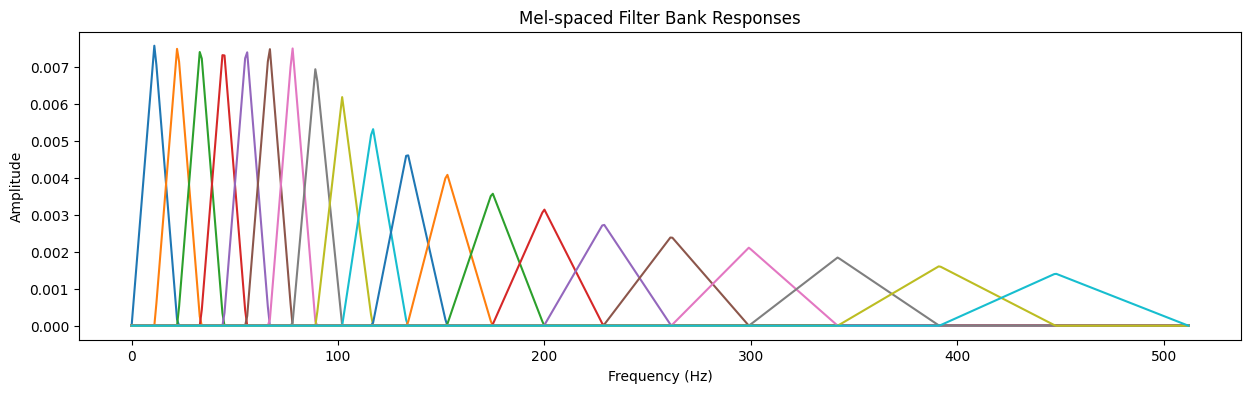




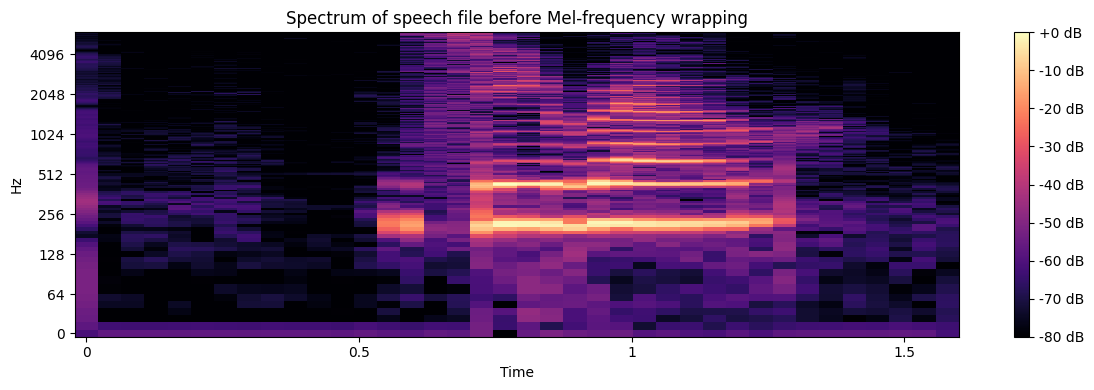
Spectrograms of this audio sample with different values of frame length (N) and overlap (M). The tradeoff between time and frequency resolution as N increases is clearly visible.

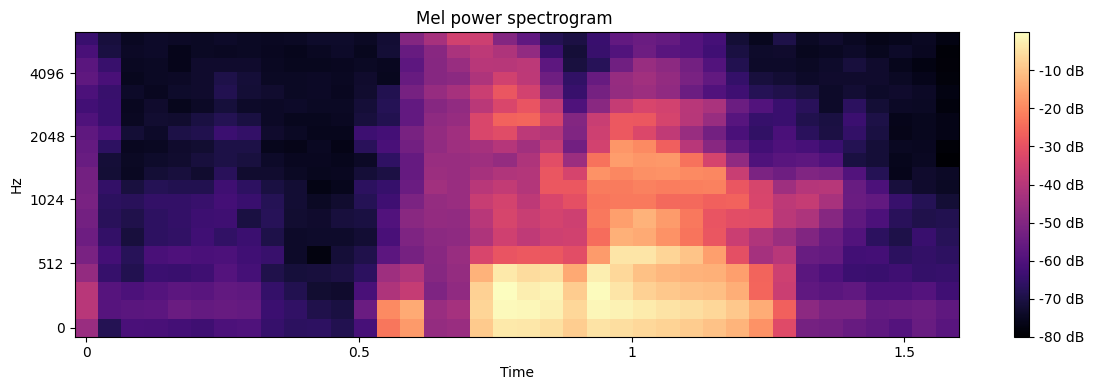
## Test 3

The Mel-spaced filter bank responses in this system utilize a linear shifting of the weights to 100 Hz rather than the standard 1 kHz. After the spacing becomes logarithmic, the height of the triangles decreases to keep total energy in each filter constant, despite the increasing bandwidth. This Mel-spaced filter bank can be seen in figure \ref{weights}.



Difference in spectrogram before and after mel-frequency wrapping step:





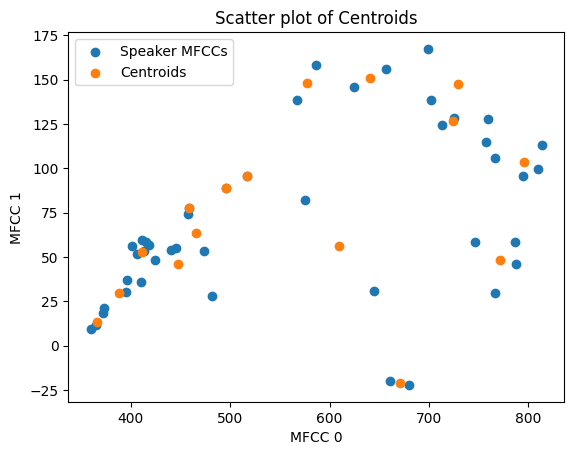
The Mel-frequency wrapping step allows a similar amount of information, from a classification standpoint, to be placed in each bin, leading to more efficient classification.

## Test 4

The feature-extraction function in our processing.py file takes in a vector of audio data and selected hyperparameters, and returns a vector of MFCC coefficients.

## Tests 5 and 6

The Tests 5 and 6 section of our Tests.ipynb file allows you to easily choose any two dimensions of the MFCCs to use as axes and plot both the MFCCs of the current audiofile, and the resultant centroids. In this case, there are 16 centroids, plotted along dimensions 0 and 1 out of the 40 MFCCs.



## Test 7

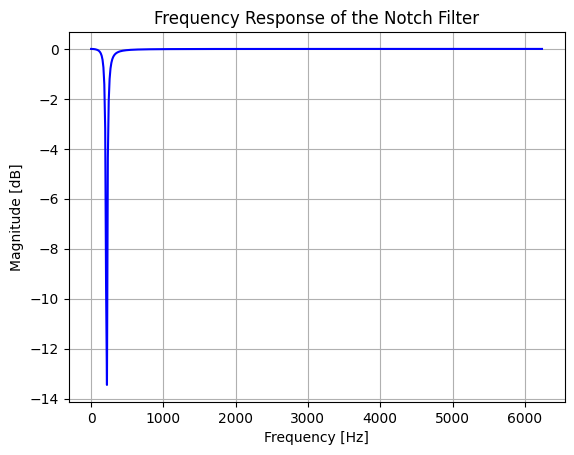
This was the baseline test of the system on the provided training and test data. It creates a codelibrary based on the training data, and then tests the 8 test data files. The system had an accuracy of 100\% with the following hyperparameters: N = 256, M = 100, n\_mfcc = 40, window = Hamming, size\_codebook = 64.

## Test 8

This test creates a notch filter at a given frequency and with a given quality factor, and then creates a new set of test files after applying the filter. The same library as Test 7 predicts these new files. The notch filter was applied to frequencies near 215 Hz, 440 Hz, 1000 Hz, and 6000 Hz. Figure \ref{notch} represents the notch filter at 215Hz.

We informally tested many different notch filters; our “official” tests consisted of notch filters as 215 Hz, 440 Hz, 1000 Hz, and 6000 Hz, each with a quality factor of 5.

This is the frequency response of the 215 Hz notch filter:



The following table summarizes our results:

|  |  |
| --- | --- |
| Notched Frequency (Hz) | Performance |
| 215 | 1.0 |
| 440 | 1.0 |
| 1000 | 0.75 |
| 6000 | 0.875 |

Overall, our system was reasonably robust in losing frequency components; the 1000 Hz range seems to have been the most key area for detection.

## Test 9

To perform this test, new training and test datasets were created, composed of the original given datasets and 10 of the “zero” in-class recordings, chosen at random. Due to the different sampling rates, 12.5 kHz for the given and 48 kHz for the in-class recordings, the given data needed to be up sampled by a factor of 4. Hyperparameters remained as 1024 and 409 for $N$ and $M$ respectively since this was the optimal parameters in the last parameter sweep. A new code library was trained on the new training dataset and the accuracy on the testing audio files was 0.83.

## Test 10

### Question 1

This test simply consisted of training and testing the system on the “twelve” in-class recordings. The accuracy of the system was found to be 100%.

### Question 2

#### a)

To perform this test, we created training and test datasets composed of every audio file: the upsampled given “zero” recordings, the “zero” in-class recordings, and the “twelve” in class-recordings. The accuracy of our system on this aggregate data-set was 0.886.

#### b)

This tested the ability of our system to distinguish not different voices, but the words “zero” and “twelve”. For a given test file, the system was considered correct as long as the predicted class was in the correct group, “zero” or “twelve”. This required a custom accuracy test specifically designed for the structure of our aggregate datasets. According to this metric, the system has an accuracy of 1.0.

# Conclusion

Overall, the system was reliable at predicting homogenous datasets. For datasets which all had the same sampling rates and background noise characteristics, the system and hyperparameters selected achieved 100% accuracy. When the data became non-homogeneous, in tests 9 and 10, performance suffered somewhat, although the accuracy remained above 80% in all cases. To improve this, a more precise rate-changing scheme might prove effective, as well as applying a filter to remove background noise. The system was robust to losing certain frequency bands, although a notch filter at 1kHz was certainly detrimental.

Other improvements which could be made are clipping the ends of the audio files, removing occasional loud background noise, and normalizing each audio file. Normalizing the energy for the audio files would reduce the influence on amplitude of the signal since loudness does not support classification.

# Contributions

Randall: Underlying class creation, parameter sweep, report outline, recorded presentation.

Conor: Class refinement, test implementation, report writing, recorded presentation.