Speech Recognition Using GMM

Group 16: Chace Nielson (30045674), Emmanuel Omari-Osei (30092729), Thanusha Veeraperumal (30090927)

Table of Contents

Introduction

Speech recognition is an innovative biometric technology which allows for the identification of individuals based on their unique vocal characteristics. This report explores the methodology and evaluation using a Gaussian Mixture Model (GMM)[1].

Speaker recognition is the process of identifying people by their voices. This innovative technology analyzes voice features and determines the speaker, like identifying a friend’s voice over the phone without visual confirmation. This is significant for several reasons: it enhances security by ensuring that only authorized voices can access services, and it enhances the personalization and user-friendliness of technology by allowing for voice commands.

Speaker recognition in this experiment relies on the Gaussian Mixture Model. This is an unsupervised learning model used to understand complex data (like voice clips) by breaking it down into simpler, understandable parts. When applied to a voice clip, Gaussian Mixture Models identify patterns and characteristics that are unique to the speaker. This is a process that involves looking at different aspects of the voice, such as pitch or tone, and understanding how these combine.

A GMM is created by feeding voice data into a model that identifies distinct parts of the data. This data can then represent the different characteristics of the speaker's voice. Once the GMM has been created for the speaker, their voice can be recognized by comparing other voice clips against their profile to check for a match.

This report will explore the use of a Gaussian Mixture model (GMM) for speaker recognition. The report will touch on the methodological framework and the analytical techniques used for performance evaluation.

Objectives

This project has two primary objectives. First, we will explore speech recognition using Gaussian Mixture Models (GMM) to analyze voice data. This will improve our understanding of how GMMs can efficiently distinguish between different speakers based on their unique voice properties. Our goal is to uncover the nuances of voice patterns and how they can be modelled to recognize individual speakers accurately.

Second, we will evaluate the performance of the GMM-based speaker recognition algorithm using a confusion matrix. This statistical tool will visualize the algorithm's performance in a comprehensible

manner. Examining the confusion matrix will give us an objective understanding of both its strengths and potential areas for improvement. Through this analysis, we will have a clearer picture of GMM's reliability and efficiency as a tool for speaker recognition.

In completing both of these objectives, we aim to provide a thorough analysis of speaker recognition using GMM, emphasizing its capabilities and laying the groundwork for further advancement in voice-based identification technologies.

Methods

The methods used for experiment design follow closely in line with the standard methodical approach for GMM training and validation. The steps for this procedure, from data acquisition to prediction, have been outlined in the section below.

Sample Acquisition

The first phase of the procedure involves the collection of audio samples from nine different subjects. Of these subjects, three were collected from our experimentation team, then another three were taken from the provided sample data, and the final three were taken from music samples, with one of the music samples distorted with noise. The audio samples taken from the subjects fall under two categories: training and testing.

Training data involved a single 30- to 60-second audio recording per speaker. Training samples required a diverse phonetic speech set within the audio to properly train the model. While this was provided for the provided sample data, we needed to use an AI-generated training speech with our experimentation team to ensure all possible phonetics were included in the speech training.

Testing data involved three to five ~10-second audio recordings per speaker. These testing samples were not constrained to include a full phonetic speech set, thus generally including random sentences and phrases. This diversity in the testing data ensures that testing reflects the natural variability in human speech.

When these files were read by the LibROSA[2] Python library they were set to use automatic sampling. This means that LibROSA determines the sampling rate from the file's metadata. This means the sample rate would be related to each audio clip.

Feature Extraction

Once the training and testing datasets are secured for each subject, the next step is focused on extracting critical speech features from the data. To perform this task, a Python audio processing library, LibROSA[2], is used to extract and calculate the Mel-Frequency Cepstral Coefficients (MFCCs) [3], a feature set encapsulating and describing the spectral envelope of an audio signal, in this case being each subject's voice. The feature extraction routine was standardized to modify the raw audio data into a structured format that fits for modelling with GMMs.

GMM Training

Following the feature extraction, the training feature sets of each subject are used to create dedicated GMMs using the ‘GaussianMixture’ module from ‘sklearn.mixture’[4]. Each GMM is designed to detect a specific subject and score the testing data based on its given subject’s training data. The number of mixture components used in the GMMs was chosen based on preliminary testing to balance model complexity with performance.

Classification

With each subject's GMM properly trained, we then move on to applying the testing dataset to the trained GMMs. In this phase, we tested data from all the subjects by having each GMM make predictions on the data. Prediction results are returned as numerical scores, which must be interpreted and classified according to the testing data chosen as the input. With this, testing data can be categorized into two types: authentic and imposter. Authentic testing data comes from the subject that the GMM is trained with and should be accepted as valid. Imposter testing data can come from any subject that the GMM has not been trained on and is expected to be rejected by the model.

After calculating the mean and standard deviation of the resulting scores, the results can be visualized as normalization curves. This allows our experimentation team to best determine any overlaps or differentiations in the scores the GMM generated for each subject.

Validation

To validate each model, we set the threshold values noted in the normalization curves generated in the Classification phase. We use critical values[5] calculated from the curves to determine the thresholds for validating each subject. This threshold is chosen with the understanding that the speaker is identified with the highest score for each testing sample. Once a threshold is set, we run more testing data against the models to see whether the models can validate the correct subjects among the set of audio samples. The intent is that the only sample to pass the validation is the subject on which the GMM is trained.

Error Rate Evaluation

Once a satisfactory threshold is chosen, the model error rates must be calculated to interpret the validity of each model. These rates are calculated with the understanding that the model should accept authentic subject audio and reject imposter subject audio. The error rates calculated are False Acceptance Rate, True Acceptance Rate, False Rejection Rate, and True Rejection Rate. Once all these error rates have been calculated, a confusion matrix[6] can be generated to visualize the model's validity, concluding the testing procedure.

Results

Nine distinct models were developed using nine different audio sources, which showed a variety of results. This section will detail the outcomes of the implemented speaker recognition system using GMM for each distinct model. These results include confusion matrix analysis, error rate evaluation and other performance metrics such as accuracy, precision and recall.

To further explore the modeling the following audio sources were used. They each contained a training clip and one or more testing clips.

Emmanuel Audio: Audio clips for Emmanuel Omari-Osei.
Chace Audio: Audio clips from Chace Nielson.
Thanusha Audio: Audio clips from Thanusha Veeraperumal.
Subjects 1, 2 and 3: Sample audio provided for the lab. Unnamed subjects speaking French.
A Jack Johnson Song: A complete song from Jack Johnson and a second song served as the training and testing data.
A Blues Song: A blues song and a different clip of the same song.
Jackhammer Distorted Audio: Training audio of someone talking and the same audio with a jackhammer playing over the top of a jackhammer noise.

The training audio clip from each source was used to create a model. This model was then compared to the testing audio for each source which included the authentic audio comparison. This led to 81 total comparisons as seen in the Jupyter Notebook After getting the comparisons the critical value threshold value was continuously tested to get the best results. For each model, 9 plots are created. 1 is for the authentic model data to be compared to the authentic testing data and the rest are imposter data.

Distribution Curves

For a detailed understanding of how each testing file related to each model, the following charts display the mean and standard deviation of the model compared to each of the 9 testing audio files. The authentic training model is compared to each training including the authentic testing. For example, the Emmanuel model is compared is Emmanual's Testing data and then 8 imposter files.

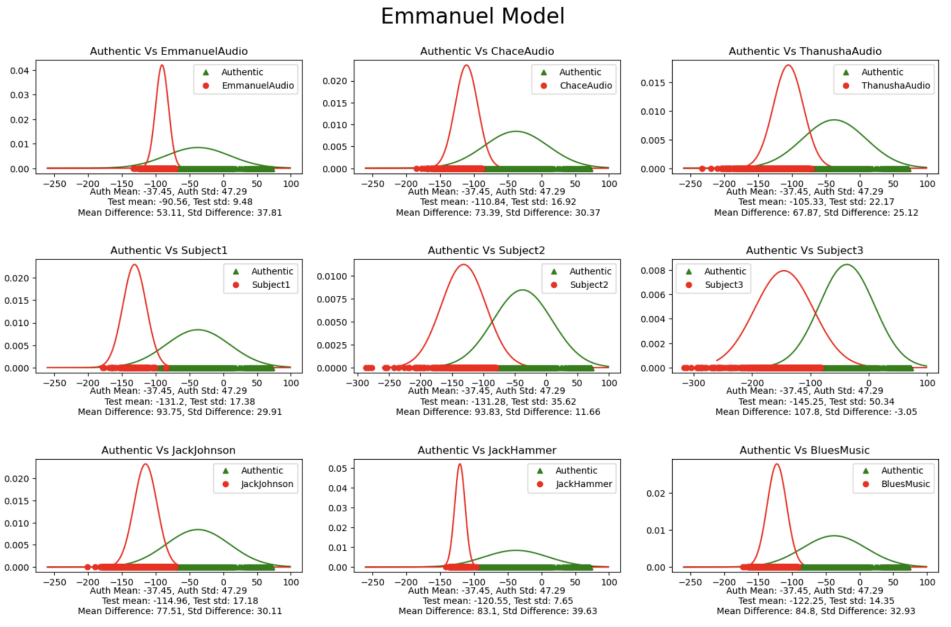


Figure #1: Gaussian Mixture Model using the authentic test file from Emmanuel and training from various other audio

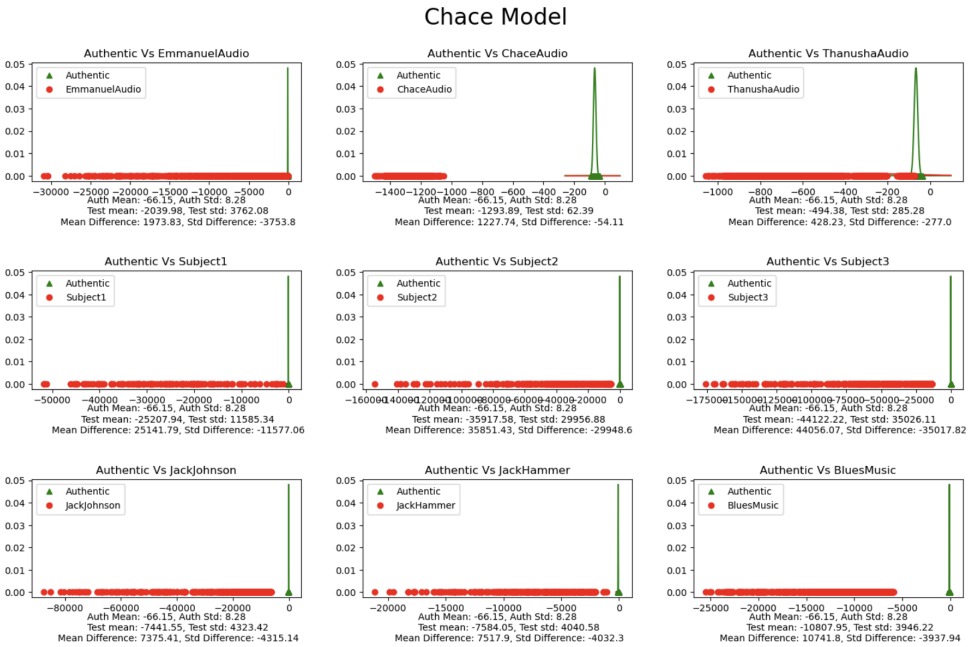


Figure #2: Gaussian Mixture Model using the authentic test file from Chace and training from various other audio

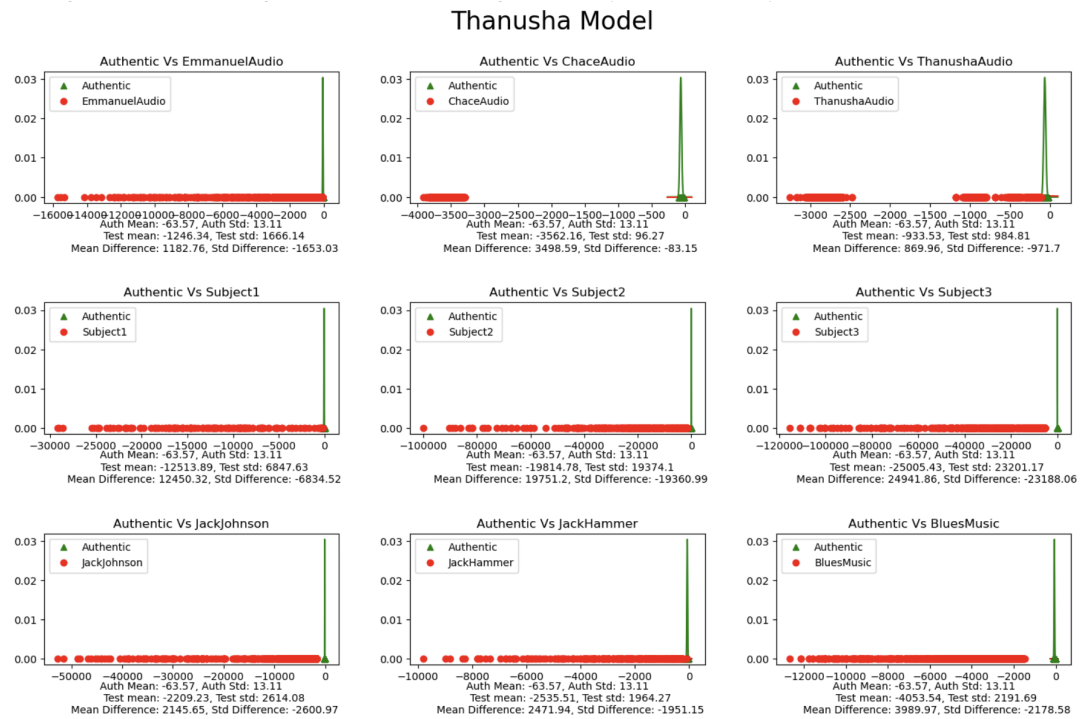


Figure #3: Gaussian Mixture Model using the authentic test file from Thanusha and training from various other audio

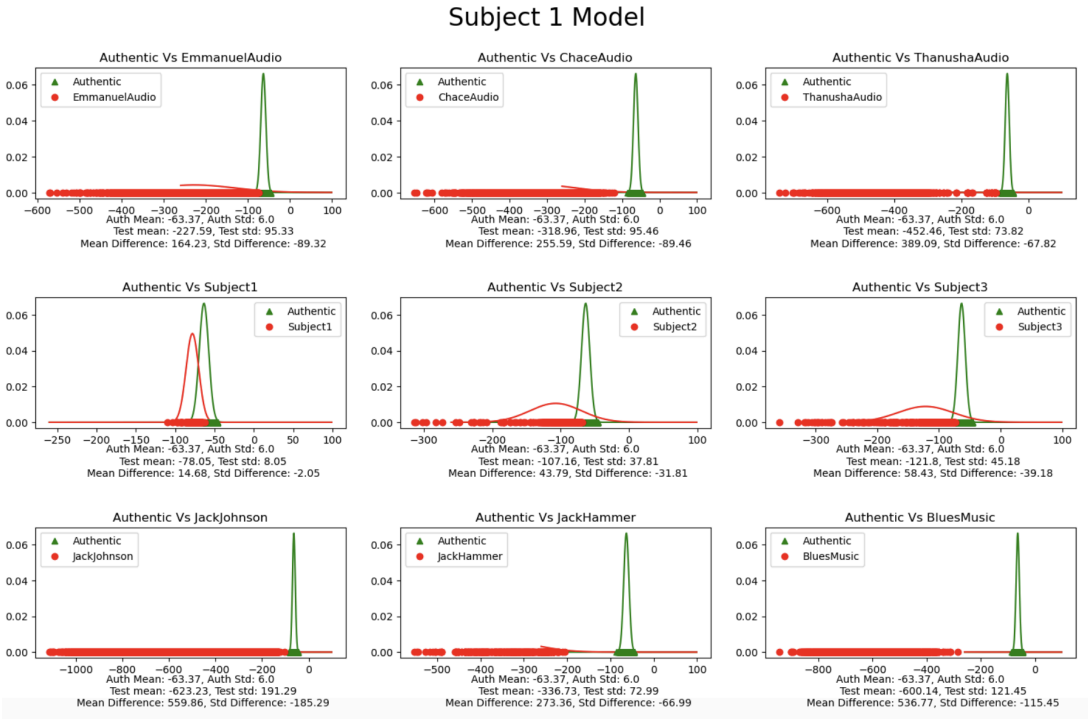


Figure #3: Gaussian Mixture Model using the authentic test file from Subject 1 and training from various other audio

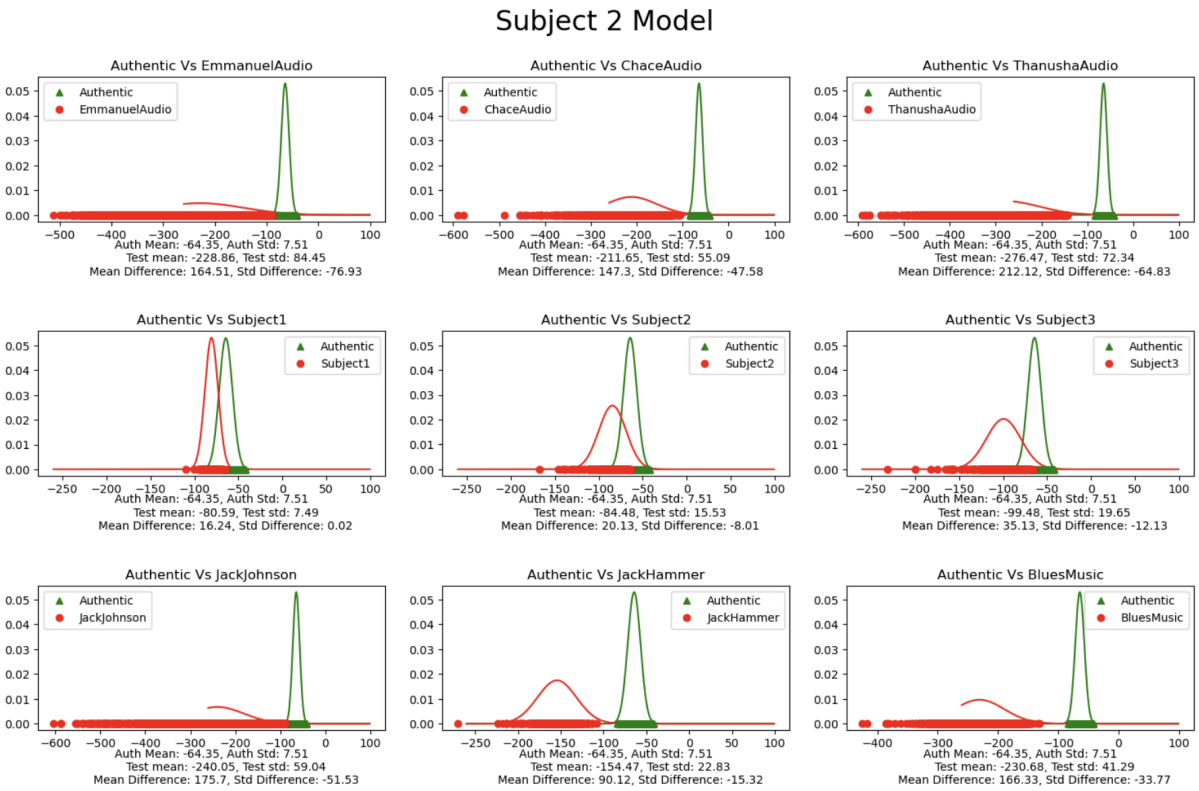


Figure #4: Gaussian Mixture Model using the authentic test file from Subject 2 and training from various other audio

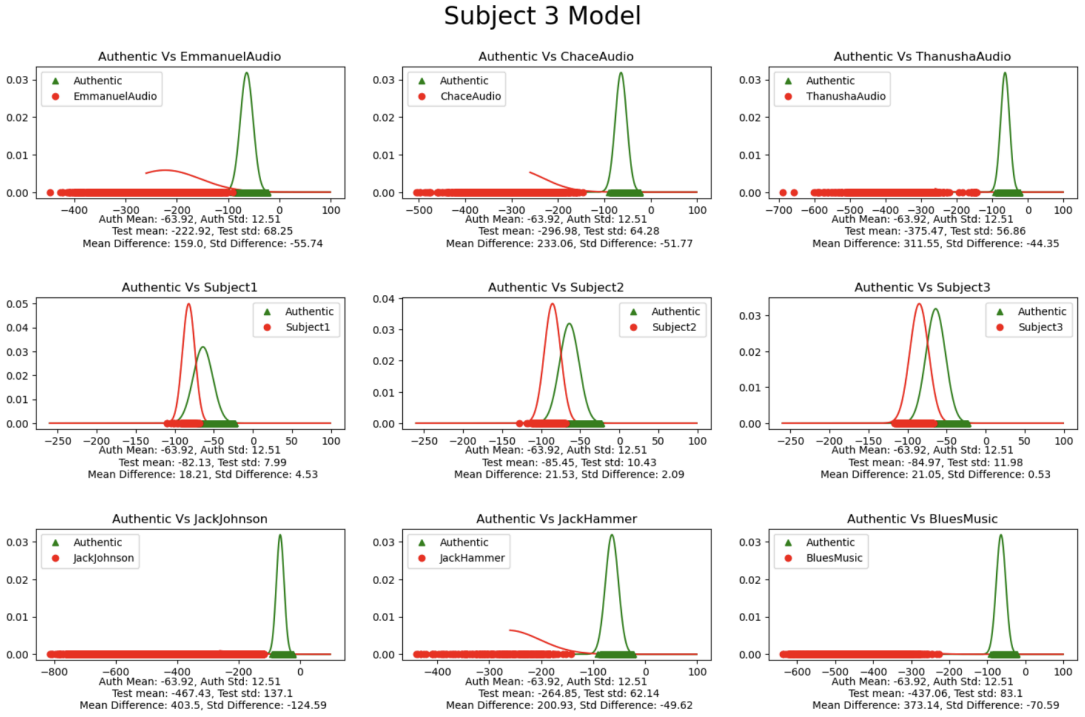


Figure #5: Gaussian Mixture Model using the authentic test file from Subject 3 and training from various other audio

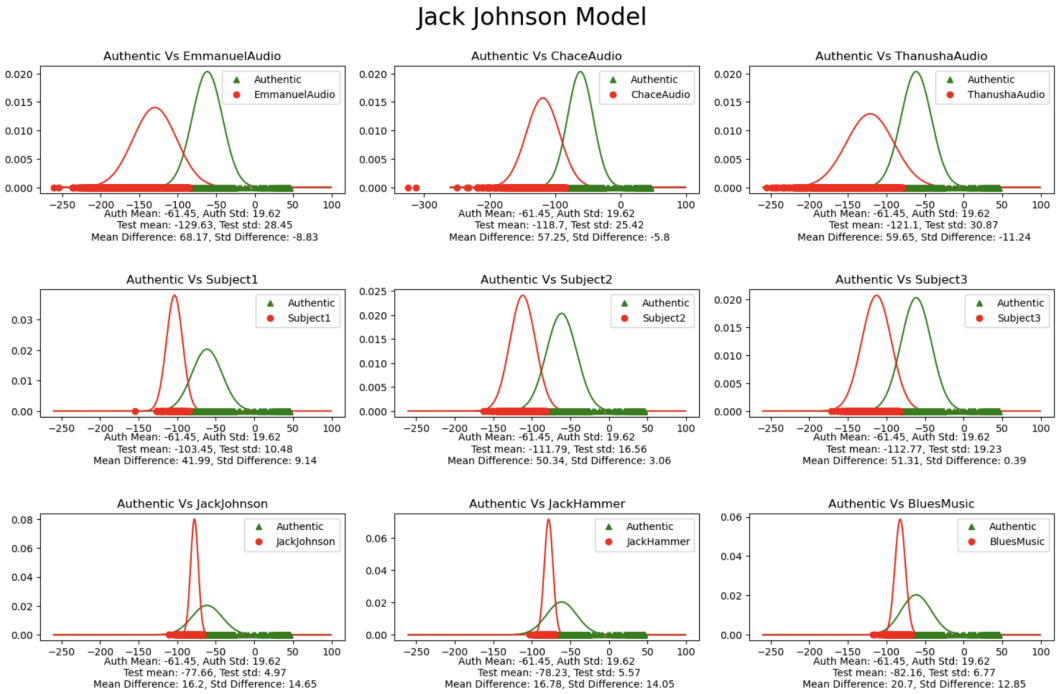


Figure #6: Gaussian Mixture Model using the authentic test file from the Jack Johnson Model and training from various other audio

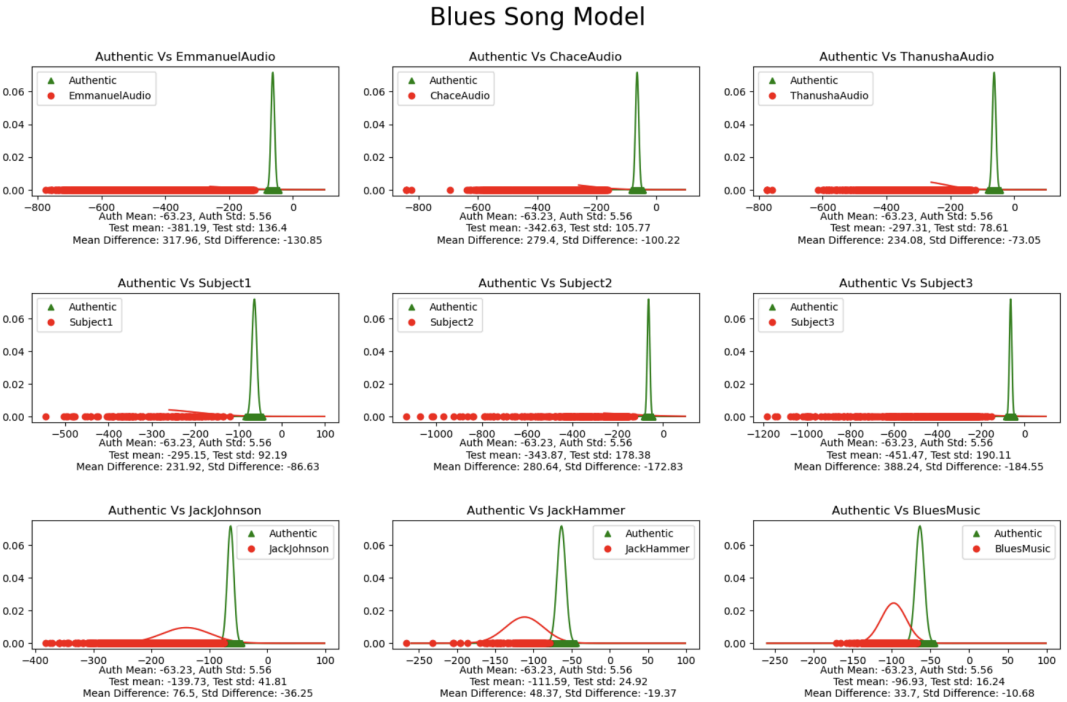


Figure #7: Gaussian Mixture Model using the authentic test file from Blue Song and training from various other audio

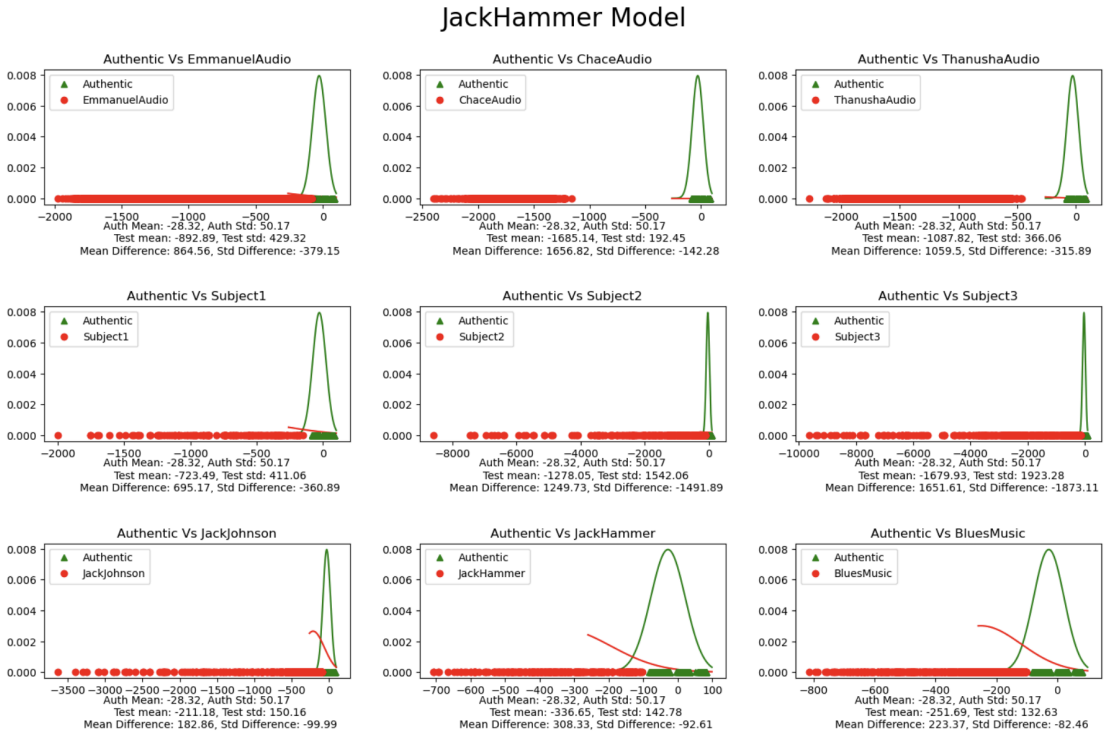


Figure #8: Gaussian Mixture Model using the authentic test file from Jack Hammer Song and training from various other audio

It can be seen that the mean of some testing scores differs drastically from the authentic training model mean score, while some are relatively close. In general, a smaller difference between the authentic training model scores and the test scores leads to a better model. Once these scores are acquired a critical value acts as a threshold to separate true from negative data which can be seen in the confusion matrix for each training model. The results from these plots will be discussed in the confusion matrix for each model below.

Confusion Matrices

The confusion matrix is used to showcase the model's ability to recognize authentic audio and reject imposter audio. This was based on a critical value score that was tuned for the best results. For each model, the confusion matrix scores are shown below. This shows the final output based on the scores shown in the plots above.

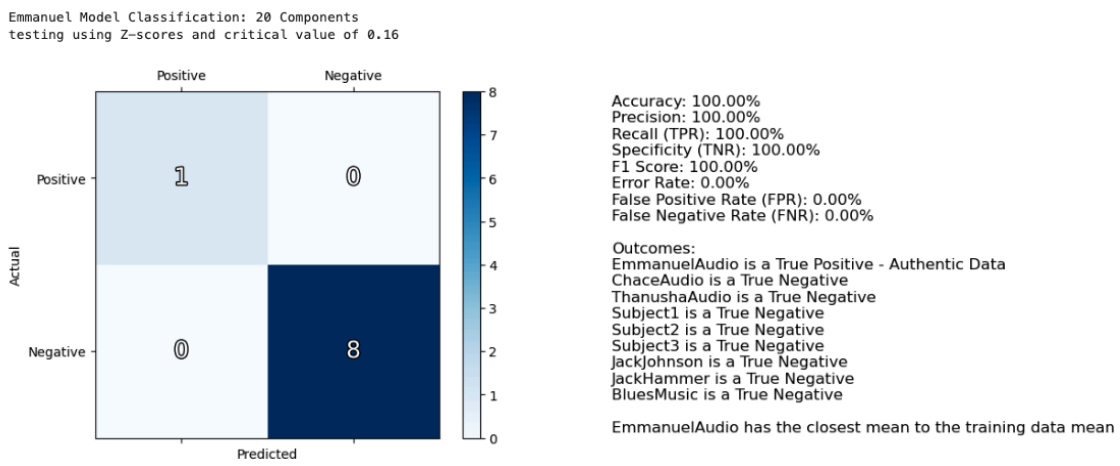


Figure #9: Confusion Matrix using Emmanuel's Model with accuracy and error percentage values

100% accuracy indicates that the training and testing data were very consistent with each other, and the features extracted for the Gaussian model captured the essence of the authentic audio very well. This allowed the critical value threshold to separate the authentic from the imposter data.

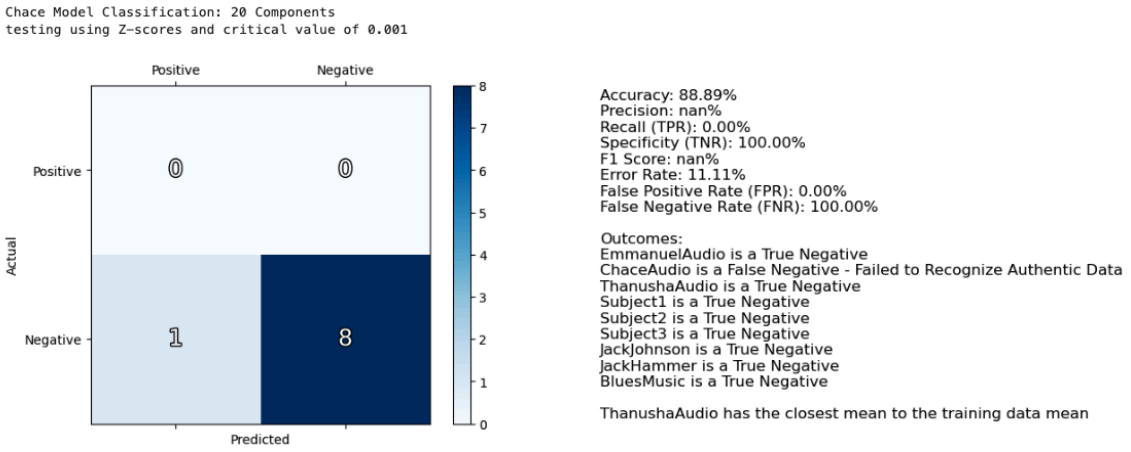


Figure #10: Confusion Matrix using Chace's Model with accuracy and error percentage values

The Chace model and Thanshua model had similar results where a critical value couldn't separate the authentic audio. Both models are discussed below as they contain similar results.

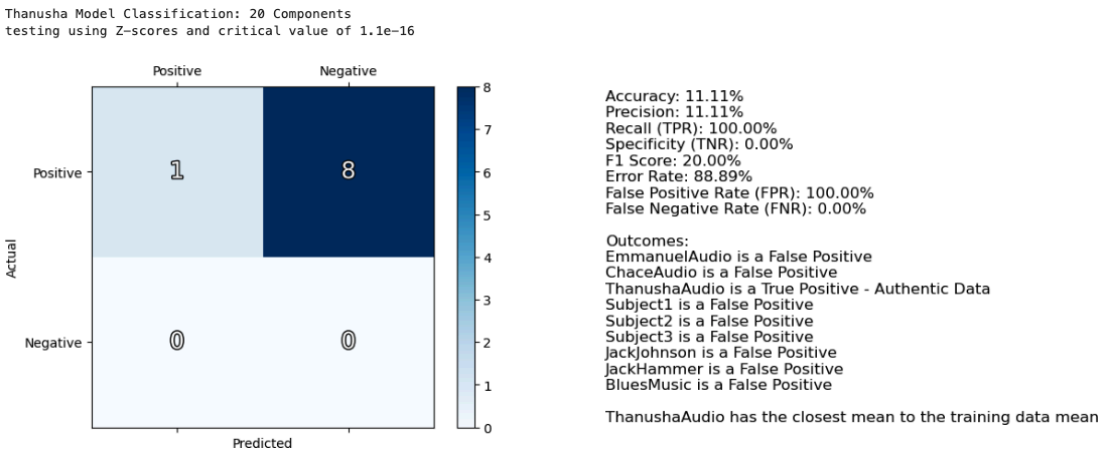


Figure #11: Confusion Matrix using Thanusha's Model with accuracy and error percentage values

These models were unable to accurately classify the audio. The critical value either failed to separate the data or was too inclusive. The exact critical value was so specific and small that it wasn't possible to find. The critical value in the Chace Model showcases a model that is too discerning and leads to only negatives. At the same time, the Thansuda model's critical value is too accepting and leads to only positives. Both of these scenarios are problems and mean that the model can't properly recognize the speech.

This might be due to the variability in the audio recordings being too high, causing overlap between the distributions of authentic and non-authentic data. Poor quality or noise in the recordings might have confused the model, and the model failed to generalize from the training data to the testing data.

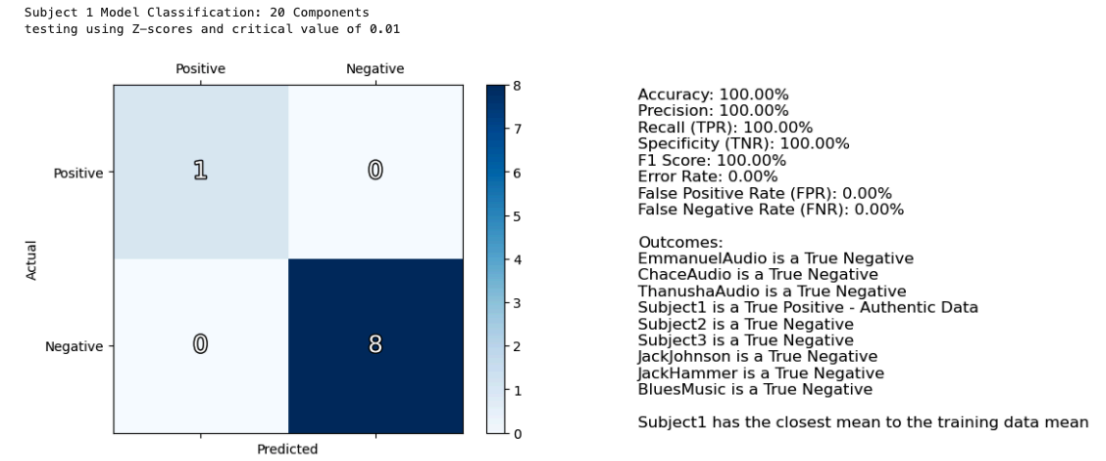


Figure #12: Confusion Matrix using Subject 1 Model with accuracy and error percentage values

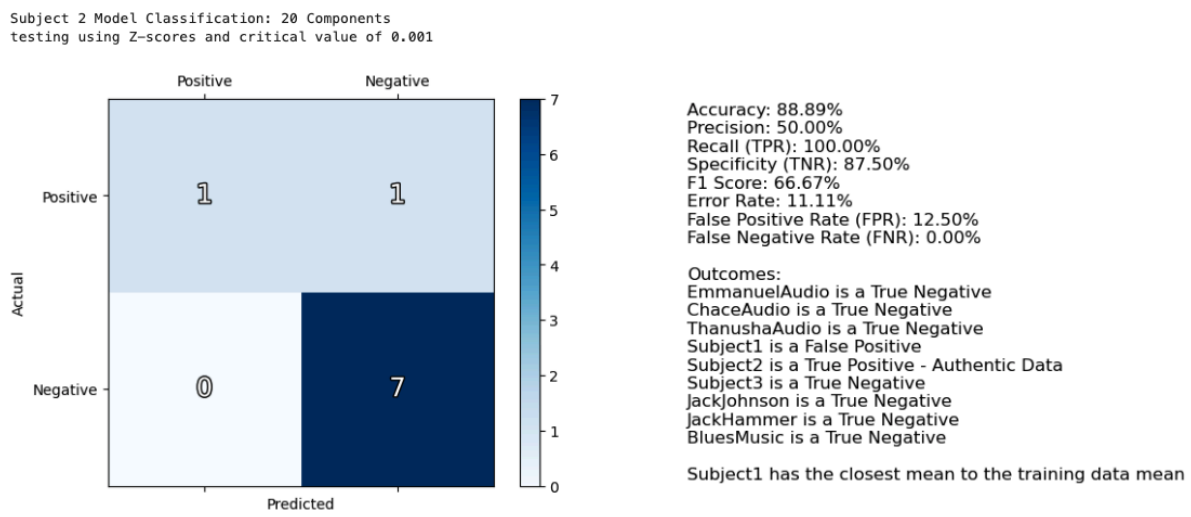


Figure #13: Confusion Matrix using Subject 2 Model with accuracy and error percentage values

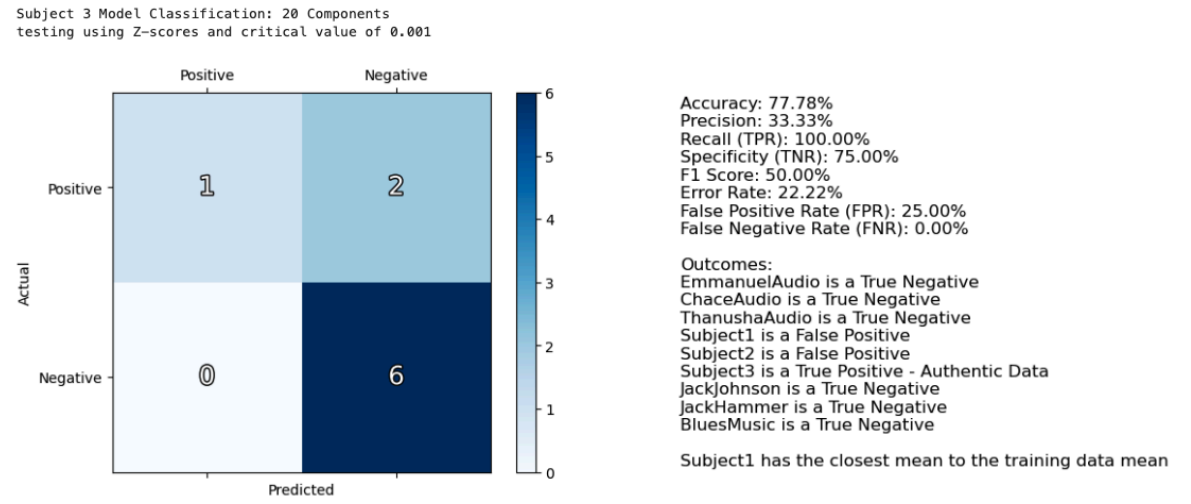


Figure #14: Confusion Matrix using Subject 3 Model with accuracy and error percentage values

Subject 1 Model: This model's performance was excellent with the lowest critical value, indicating a good fit between the model and the data. The critical value was capable of separating the data and obtaining just the authentic data as a true positive.

Subject 2 Model: The model resulted in one false positive, which suggests that the critical value or the model's sensitivity may need slight adjustment.

Subject 3 Model: With two false positives, this model appears to be less discriminating than the Subject 1 Model and might benefit from re-evaluation of the critical value or additional training data for the subject.

It's possible that the audio for the sample subjects was all very similar which led to false positives in the models for subjects 2 and 3. Other testing data was distinct enough to be differentiated from the training model.

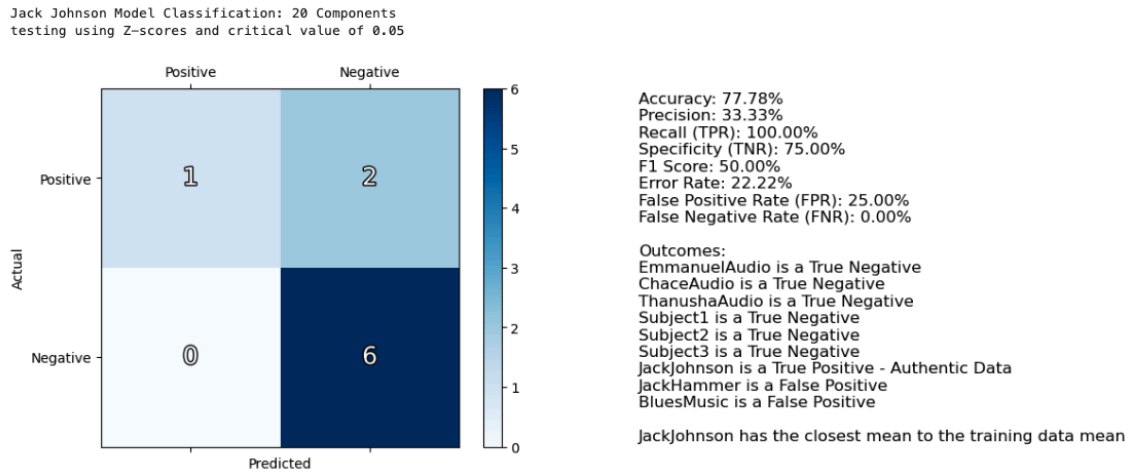


Figure #15: Confusion Matrix using Jack Johnson Model with accuracy and error percentage values

This audio was a test in addition to the normal speech. We wanted to find out if music could be recognized. False positives came from the other testing song and audio with distortion which may have a similar effect as background music. This may indicate that the audio features used by the model might be capturing more generic music characteristics rather than specific patterns unique to the training data.

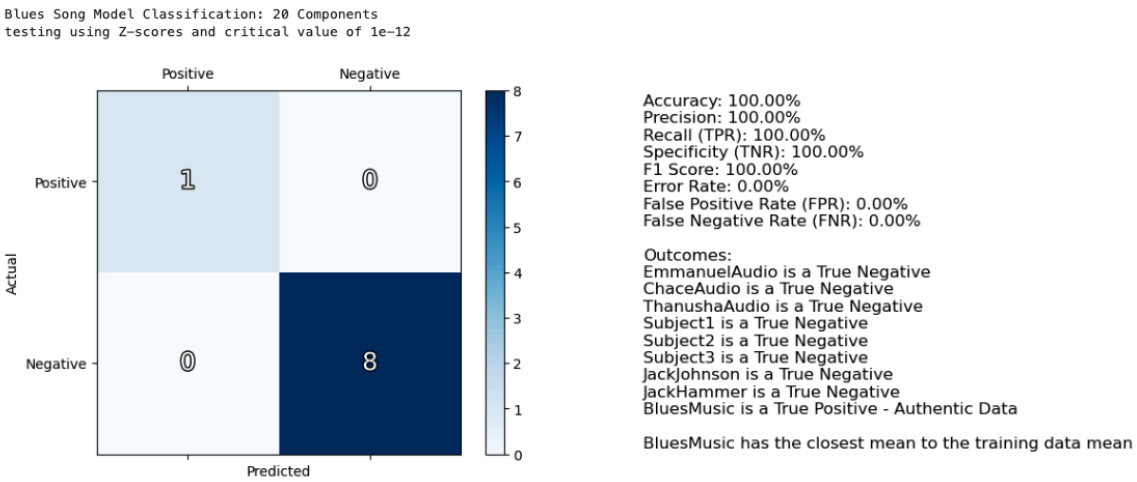


Figure #16: Confusion Matrix using Blue Song Model with accuracy and error percentage values

Perfect accuracy with a tight critical value suggests that different sections of the same song have enough unique characteristics that the model can use to distinguish between them and differentiate between other songs.

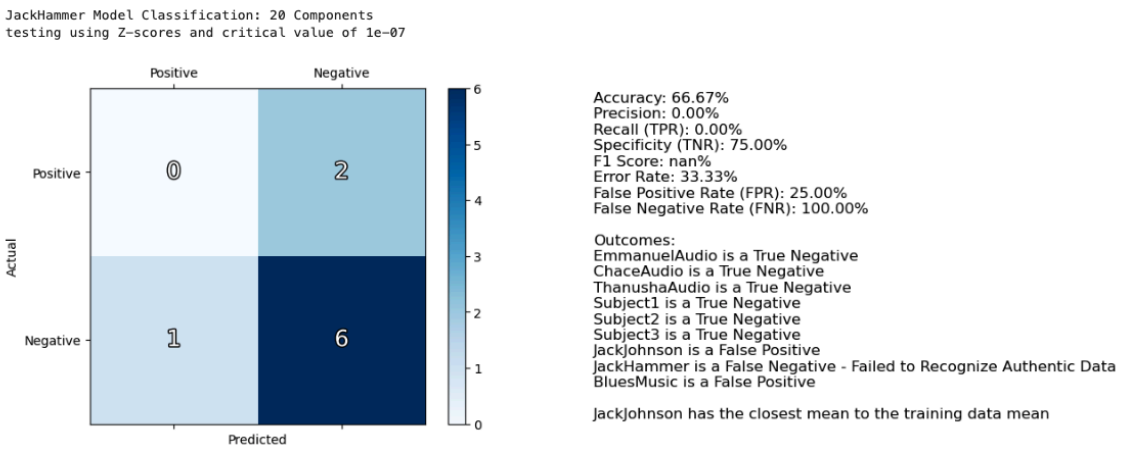


Figure #17: Confusion Matrix using Jack Hammer Model with accuracy and error percentage values

This model's goal was to discern if overlaid noise (jackhammer sounds) could still allow for recognition of the original audio. The jackhammer distortion prevented the audio from being recognized although the music data was recorded as a false negative meaning the jackhammer audio might have been recognized as similar to the jackhammer noise by the model.

As a last note, it was found that the number of Gaussian components didn't have a significant impact on performance. This led us to pick 20 components as the number used for all models to maintain consistency.

Applications

Voice Authentication: The models could be used for biometric security systems that recognize individuals based on their voice patterns.

Forensic Analysis: In law enforcement, the technology could help validate the authenticity of audio recordings in investigative work.

Speech Recognition: The models could enhance speech recognition software, improving voice command systems. This could include integration with smart devices for better recognition of different household members, and customizing responses and actions accordingly

Music Identification: Similar to how Shazam works, your models could be used to identify songs or determine

the genre of music from audio samples.

Multimedia Management: Assisting in organizing and searching through large audio databases by identifying and categorizing content.

Language Learning Tools: Providing feedback on pronunciation and accent in language learning applications.

Conclusion

This report has explored the concept of speaker recognition using Gaussian Mixture Models (GMMs) and its ability to recognize authentic testing data for various training models. The goals were to learn more about its effectiveness, applications, and limitations.

Through the approach outlined in the Methods section, audio samples were collected, speech features extracted, GMMs trained, classification performed, and finally, results were recorded using various metrics, including confusion matrices. Our exploration led to several significant findings and insights.

Firstly, we observed the effectiveness of GMM-based speaker recognition, exemplified by the high accuracy achieved in certain models. Models like the Subject 1 Model and Emmanuel Model showcased excellent performance, accurately distinguishing between authentic and imposter audio samples. However, challenges arose with models like the Chace and Thanusha models, where the critical value struggled to differentiate between authentic and imposter data, indicating the need for further refinement in those cases.

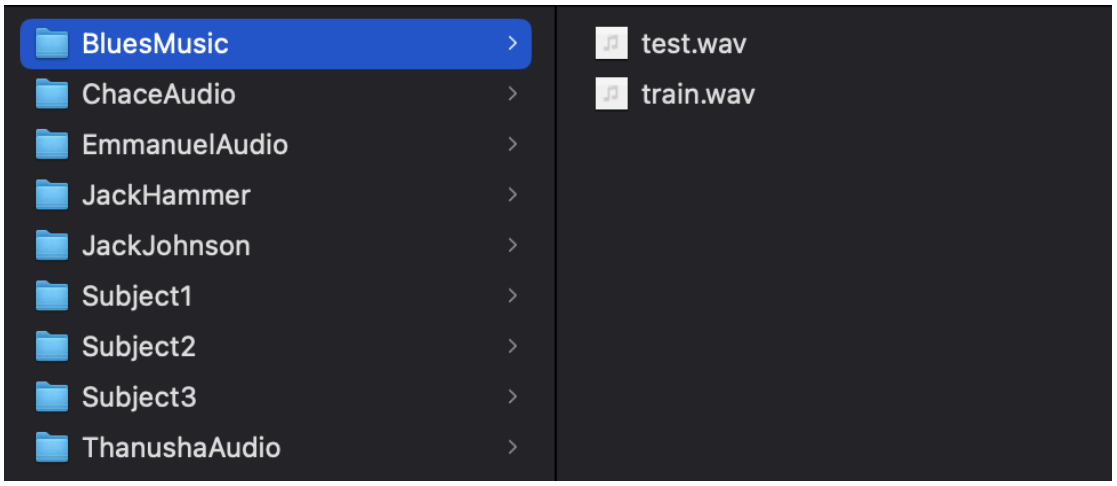
Furthermore, we identified the variability in speech as a significant challenge in audio classification. Unlike more static biometrics, such as fingerprints or signatures, human voice and music audio recordings have large fluctuations in tone, pitch, and background noise, adding complexity to the classification process. This variability displays a need for high-quality, consistent audio data for reliable classification and even for testing data.

There are several avenues for improvement. Fine-tuning the critical value threshold and exploring advanced feature extraction techniques could enhance the model's ability to differentiate between authentic and imposter audio. Additionally, incorporating deep learning approaches or ensemble methods may offer further improvements in accuracy and robustness. This combined with higher-quality audio samples could vastly improve speech recognition.

Appendix

Instructions for Running the Code

- Download the Jupyter Notebook and the folder containing all the data labelled “AudioFiles”.
- Make sure the notebook and data file are in the same directory
- The Jupyter Notebook will now access the data files and you can run through the code in Juypter Notebook.



The folder “AudioFiles” contains the train and test audio files. These files are accessed by the code.

Code Snippets

The code is broken up into functions which can be called for each model's training data and be compared to any amount of testing data. This design allowed the code to be modular and provided freedom for adding new data and testing new models. A few of the important functions are shown below.

Data Setup

```
# set up files for various tests using  different users as training and for validation and testing
Emmanuel_train_files = get_files("EmmanuelAudio", "train")
Emmanuel_test_files = get_files("EmmanuelAudio", "test")

Chace_train_files = get_files("ChaceAudio", "train")
Chace_test_files = get_files("ChaceAudio", "test")

Thanusha_train_files = get_files("ThanushaAudio", "train")
Thanusha_test_files = get_files("ThanushaAudio", "test")

Subject1_train_files = get_files("Subject1", "train")
Subject1_test_files = get_files("Subject1", "test")

Subject2_train_files = get_files("Subject2", "train")
Subject2_test_files = get_files("Subject2", "test")

Subject3_train_files = get_files("Subject3", "train")
Subject3_test_files = get_files("Subject3", "test")

JackJohnson_train_files = get_files("JackJohnson", "train")
JackJohnson_test_files = get_files("JackJohnson", "test")

JackHammer_train_files = get_files("JackHammer", "train")
JackHammer_test_files = get_files("JackHammer", "test")

BluesMusic_train_files = get_files("BluesMusic", "train")
BluesMusic_test_files = get_files("BluesMusic", "test")

# set up imposters for all the tests - will also contain authentic testing data
imposter_tests = [
                Emmanuel_test_files,
                Chace_test_files,
                Thanusha_test_files,
                Subject1_test_files,
                Subject2_test_files,
                Subject3_test_files,
                JackJohnson_test_files,
                JackHammer_test_files,
                BluesMusic_test_files
                ]
imposter_names = [
                "EmmanuelAudio",
                "ChaceAudio",
                "ThanushaAudio",
                "Subject1",
                "Subject2",
                "Subject3",
                "JackJohnson",
                "JackHammer",
                "BluesMusic",
                ]
```

This code is a data setup for the training and testing files used for multiple training models. The names are used to provide more context for each testing result. The code could be altered to include more testing data in the imposter tests list. Using the runAnalysis function shown below the code could also include multiple authentic tests as long as the indices of the authentic tests in the imposter list are made known in the call to runAnalysis. Its important that the imposter_tests and imposter_names lists are the same length.

Running analysis for training and testing sets

```
# auth index is the index of the impostor list that contains authentic data for ease of running
def run_analysis(databaseUser, training_files, imposter_list, imposter_names, auth_index, critical_value, NUMGCOMPONENTS=20):

    # get features for training, authentic, and all imposters
    training_mfccs = extract_mfcc_features(training_files)

    impostor_features_list = []
    for imposter in imposter_list:
        impostor_features_list.append(extract_mfcc_features(imposter))

    # train the gmm
    gmm_model = train_gmm(training_mfccs, NUMGCOMPONENTS)

    auth_test_scores, auth_labels = test_data_against_gmm(gmm_model, training_mfccs)

    imposter_scores_list = []
    imposter_labels_list = []

    for imposter_features in impostor_features_list:
        score, label = test_data_against_gmm(gmm_model, imposter_features)
        imposter_scores_list.append(score)
        imposter_labels_list.append(label)

    # now we have the scores for authentic data and for multiple impostor data
    x = np.arange(-260, 100, 1)

    # Calculate Mean and Standard Deviation of Authentic Test Scores
    auth_Mu = np.mean(auth_test_scores)
    auth_Std = np.std(auth_test_scores)
    auth_Prob = norm.pdf(x, loc=auth_Mu, scale=auth_Std)

    # Calculate Mean and Standard Deviation of each of the imposters
    imposter_mean_list=[]
    imposter_std_list=[]
    imposter_prob_list=[]

    for imposter_scores in imposter_scores_list:
        # Calculate Mean and Standard Deviation of Imposter Test Scores
        imp_mu = np.mean(imposter_scores)
        imp_std = np.std(imposter_scores)
        imp_Prob = norm.pdf(x, loc=imp_mu, scale=imp_std)

        # add to list
        imposter_mean_list.append(imp_mu)
        imposter_std_list.append(imp_std)
        imposter_prob_list.append(imp_Prob)
```

```
    # Initialize counters
    TP = FP = TN = FN = 0

    # Initialize a list to hold the outcome strings
    outcomes = []

    for i, impostor in enumerate(imposter_names):
        # get the mean of the imposter and then use it to compare
        test_mu = imposter_mean_list[i]
        match_value = test_zscore(test_mu, auth_Mu, auth_Std, critical_value)
        if i == auth_index:  # Assuming auth_index is the index of the authentic case
            if match_value:
                outcomes.append(f"{impostor} is a True Positive - Authentic Data")
                TP += 1
            else:
                outcomes.append(f"{impostor} is a False Negative - Failed to Recognize Authentic Data")
                FN += 1
        else:
            if match_value:
                outcomes.append(f"{impostor} is a False Positive")
                FP += 1
            else:
                outcomes.append(f"{impostor} is a True Negative")
                TN += 1
```

The rest of the runAnalysis function is responsible for plotting charts and the confusion matrix

Below is the line of code to run a single model and compare it to multiple testing sets. The auth_index argument is the position of authentic data in the imposter list for results. It can also be a list if multiple authentic data sets are used for testing example auth_index=[0,3,4] .

```
print("Training data is Emmanuels training data and its authentic testing data is multiple shorter audio clips.")
run_analysis("Emmanuel Model",Emmanuel_train_files, imposter_tests, imposter_names, auth_index=0, critical_value=0.16)
```

```
Training data is Emmanuels training data and its authentic testing data is multiple shorter audio clips.
```

Other important functions

```python
# returns true of false based on the critical value
def test_zscore(test_audio_mean, real_mean, real_std, critical_value=0.05):
    """
    Classify a test signature as genuine or impostor based on a critical value threshold.

    Parameters:
    test_sig (float): The signature score to be tested.
    real_mean (float): The mean score of genuine signatures.
    real_std (float): The standard deviation of genuine signatures.
    critical_value (float): The critical value for determining the threshold.

    Returns:
    bool: True if the signature is classified as genuine, False otherwise.
    """
    # Calculate the Z-score for the given critical value
    z_score = norm.ppf(1 - critical_value/2)  # Two-tailed test

    # Set the threshold based on the Z-score
    lower_bound = real_mean - z_score * real_std
    upper_bound = real_mean + z_score * real_std

    return lower_bound <= test_audio_mean <= upper_bound
```

```python
# this function take a model and test samples and returns there scores and labels
# the labels are not used as a different methods to find the predictions and labels was used
def test_samples(gmm_model, test_samples):
    # Make array lists for predictions and scores
    test_scores = []
    prediction_labels = []

    # Iterate through test samples:
    for test_sample in test_samples:

        # Make predictions on test sample
        prediction_label = gmm_model.predict(test_sample)

        # Collect prediction scores
        test_score = gmm_model.score_samples(test_sample)

        # Append to result lists
        test_scores.append(test_score)
        prediction_labels.append(prediction_label)

    return test_scores, prediction_labels
```

```python
# this takes the socres and flattens then to be returned to calling function
def test_data_against_gmm(gmm_model, test_data_freatures):
    # store the scores
    scores = []

    # test the sample data
    test_scores, prediction_labels = test_samples(gmm_model, test_data_freatures)

    # Flatten score and prediction lists
    flat_test_scores = [i for ix in test_scores for i in ix]
    flat_prediction_labels = [i for ix in prediction_labels for i in ix]

    return flat_test_scores, flat_prediction_labels
```

```python
# Define path to the data
data_root_folder = 'AudioFiles'

# Function to get file paths for a subject
def get_files(subject, set_type):
    subject_folder = os.path.join(data_root_folder, subject)
    files = os.listdir(subject_folder)
    return [os.path.join(subject_folder, file) for file in files if set_type in file]
```

```python
# Function to extract MFCC features and return the array of features
def extract_mfcc_features(file_paths):
    mfcc_features = []
    for file_path in file_paths:
        # Load audio file
        audio_signal, sample_rate = librosa.load(file_path, sr=None)
        # Extract MFCC features
        mfcc = librosa.feature.mfcc(y=audio_signal, sr=sample_rate)
        mfcc_features.append(mfcc.T)
    return mfcc_features
```

```python
# This function trains a gmm model using a default 20 Gaussian components using training data and returns the Model
# it assume the training data is a list of features
def train_gmm(training_data, NUMGCOMPONENTS=20):
    # Create the Gaussian Mixture Model
    gmm = GaussianMixture(n_components=NUMGCOMPONENTS)

    # Perform the training using the training data samples
    for sample in training_data:
        gmm.fit(sample)

    return gmm
```

Other functions (not shown) include plotting functions with Matplotlib. They are used to convert the scores and results into plots and graphs seen in the report.

References

[1] A. Singh, "Build Better and Accurate Clusters with Gaussian Mixture Models," Analytics Vidhya, Dec. 13, 2023. [Online]. Available: https://www.analyticsvidhya.com/blog/2019/10/gaussian-mixture-models-clustering/.

[2] B. McFee et al., "LibROSA 0.9.0," 2022. [Online]. Available: https://librosa.org/doc/latest/index.html.

[3] E. Deruty, "Intuitive understanding of MFCCs," Medium, Sep. 16, 2022. [Online]. Available: https://medium.com/@derutycsl/intuitive-understanding-of-mfccs-836d36a1f779.

[4] "sklearn.mixture - Gaussian Mixture Models (GMM)" Scikit-learn: Machine Learning in Python, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html.

[5] "What is a z-score? What is a p-value?" ArcGIS Pro Documentation, [Online]. Available: https://pro.arcgis.com/en/pro-app/3.1/tool-reference/spatial-statistics/what-is-a-z-score-what-is-a-p-value.html.

[6] Sarang Narkhede, "Understanding Confusion Matrix," Towards Data Science, May 9, 2018. [Online]. Available: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.