University of Tehran

Gender Classification and Speaker Authentication

Amirabbas Reza Soltani - 610399205

Arad Vazirpanah - 610399182

Contents

1 Abstract

The project involves applying machine-learning techniques to the analysis of speech signals, which have two main tasks: gender classification and voice authentication. It aims at designing and developing models that can classify the gender of a speaker accurately and authenticate or identify a person from a predefined set of speakers. This piece of work bridges the gap between theoretical understanding and practical implementation by using state-of-the-art methods in signal processing and feature extraction to overcome inherent challenges in audio data analysis. The project shows how machine learning can be an effective approach to solving real-world problems in voice recognition and classification.

2 Introduction

2.1 Voice Authentication

Voice Authentication refers to verifying a person's identity based on their voice patterns. It works by analyzing unique vocal characteristics, such as pitch, tone, and cadence, which differ for every individual.

2.1.1 Importance

Voice authentication is crucial for secure, contactless identity verification and is widely adopted in fields such as banking, healthcare, and access control. Its ease of use and natural interaction make it ideal for modern user authentication systems.

2.1.2 Applications

Voice authentication has many applications. Two most important application are:

- **Speaker Identification**: Determines "who is speaking" from a pool of known speakers. Used in scenarios like customer service or security systems.
- **Gender Classification**: Detects the speaker's gender using voice features. Applications include demographic studies, marketing analysis, and tailored user experi-

ences.

2.1.3 Types of Authentication

Authentication systems can be categorized into two main types based on the scope and requirements of identifying speakers:

- **Closed-set Authentication**: In this type of authentication, the system assumes that the speaker is one of the predefined and finite set of users. The main objective here is to correctly identify the speaker among known individuals in the system. Closed-set authentication is typically employed in controlled environments where the list of authorized users is fixed and well-defined. The performance of such systems heavily relies on the quality of models trained on sufficient and representative data for each user.

  Examples of this type of authentication include:

  - *Biometric access control for a small team or organization.*
  - *Speaker identification in secure facilities where only registered individuals are allowed.*

  While closed-set systems are effective in restricted domains, they are not designed to handle unknown speakers, which limits their flexibility in broader, dynamic environments.

- **Open-set Authentication**: Unlike closed-set systems, open-set authentication does not assume that all speakers are from a predefined set. The system must determine whether the speaker is known (authorized) or unknown (unauthorized). This adds an additional layer of complexity as the system must not only identify authorized users but also reject unauthorized ones.

  Open-set systems face challenges such as:

  - Handling variations in speech patterns, accents, and noise for both authorized and unauthorized speakers.

  - Balancing between false acceptances (unauthorized individuals being accepted) and false rejections (authorized individuals being denied).

Applications of open-set authentication are broader and more dynamic, including:

  - *Voice authentication for customer services accessible to the public.*
  - *Fraud detection in online banking or phone-based transactions.*
  - *Access control for systems with an ever-changing set of users, such as cloud platforms.*

Open-set systems are particularly useful in scenarios where new or unauthorized users are expected, but they require more advanced algorithms and robust training data to manage the variability.

2.1.4 Voice Authentication Challenges

Voice authentication systems face several challenges that can affect their accuracy, robustness, and applicability in real-world scenarios. Some of the key challenges, along with potential solutions, are outlined below:

- **Noise Interference**: Environmental noise, such as background chatter, mechanical sounds, or echoing, often disrupts the clarity of voice signals. This interference reduces the effectiveness of feature extraction and impacts the model's accuracy.

  Potential solutions:

  - **Noise Reduction Algorithms**: Techniques like spectral subtraction, Wiener filtering, and adaptive filtering can remove or suppress background noise. These methods are particularly effective when applied during the preprocessing stage.
  - **Noise-Canceling Hardware**: Using directional microphones, noise-canceling headsets, or acoustic isolation techniques can help capture cleaner voice signals, minimizing the need for extensive software-based noise suppres-

sion.

  - **Voice Activity Detection (VAD)**: Implementing VAD to separate speech segments from silence or noise can enhance signal clarity before further processing.

- **Speaker Variability**: Individual voice characteristics can vary significantly due to factors such as age, health, emotional state, or even external conditions like microphone type and recording environment. This variability makes it challenging to create generalized models.

  Potential solutions:

  - **Diverse Training Data**: Collect and use datasets that include a wide range of voice samples, covering variations in age, gender, accent, and recording environments to improve the model's robustness.
  - **Robust Feature Extraction**: Extract invariant and reliable features, such as MFCCs or PLPs (Perceptual Linear Prediction), which remain effective across diverse conditions.
  - **Transfer Learning**: Use pre-trained models on large, diverse datasets to fine-tune on specific tasks, enabling better adaptation to speaker variability.

- **Gender Classification Challenges**: Some vocal features, like pitch or spectral properties, often overlap between males and females, making gender classification challenging, especially in ambiguous cases or for speakers with atypical voice ranges.

  Potential solutions:

  - **Advanced Deep Learning Models**: Utilize CNNs or recurrent neural networks (RNNs) to capture complicated patterns in voice signals and improve feature discrimination between genders.
  - **Feature Engineering**: Combine spectral features (e.g., MFCCs, log Mel spectrograms) with temporal features (e.g., zero-crossing rate, energy) to en-

hance classification accuracy.

  - **Multi-Task Learning**: Design models that simultaneously perform gender classification and other related tasks, leveraging shared information to improve performance.

- **Data Scarcity**: Labeled datasets for voice authentication and gender classification are often limited, particularly for underrepresented groups or specific accents and conditions.

  Potential solutions:

  - **Data Augmentation**: Apply techniques like adding background noise, pitch shifting, speed adjustment, or time stretching to artificially increase the diversity of training data. [8]
  - **Synthetic Voice Generation**: Use tools like text-to-speech synthesis or voice cloning technologies to generate additional samples for underrepresented demographics. [16]

2.2 Preprocessing of Audio Data

Preprocessing plays a vital role in ensuring the accuracy and reliability of machine learning models for voice authentication and gender classification. Raw audio signals are often noisy, inconsistent, and contain redundant or irrelevant information. Without proper preprocessing, the performance of machine learning models can degrade significantly.

- **Noise Reduction**: Audio signals are often contaminated by environmental noise, such as background conversations, machinery sounds, or echoes. Noise reduction techniques, such as spectral subtraction, band-pass filtering, or Wiener filtering, help remove unwanted components, ensuring cleaner signals for feature extraction. [11] Common techniques include:

  - **Spectral Subtraction**: Removes noise by estimating it from silent portions of the recording.

  - **Wiener Filtering**: Dynamically adjusts noise reduction based on frequency components.

- **Normalization**: Speech signals can vary in amplitude and normalization ensures that the signal's amplitude is scaled uniformly, making it easier for models to focus on the features that truly matter, such as pitch or formants (Formants are frequency peaks in the spectrum which have a high degree of energy). [9]

- **Windowing**: Speech is non-stationary, meaning its properties change over time. Windowing splits audio signals into overlapping segments (windows) to facilitate feature extraction by analyzing temporal features, such as energy variations and pitch contours, more effectively. Common windows include Hamming or Hanning windows, which reduce edge artifacts. [5]

2.3 Feature Extraction Techniques

To analyze audio signals effectively, various features are extracted that represent different characteristics of the sound. These features enable machine learning models to better understand and classify audio data. Below are the most commonly used feature extraction techniques and a brief explanation of their algorithms:

- **MFCC (Mel Frequency Cepstral Coefficients)**: MFCCs capture the shape of the vocal tract, which is critical for speech processing. [2] The algorithm involves:
  - Dividing the audio into overlapping frames.
  - Applying the Fast Fourier Transform (FFT) to convert each frame into the frequency domain.
  - Passing the frequency data through a bank of Mel filters, which mimic the human ear's perception of sound.
  - Taking the logarithm of the filter bank energies to compress dynamic ranges.
  - Applying the Discrete Cosine Transform (DCT) to decorrelate the Mel energies, resulting in compact coefficients.

MFCCs are widely used because they provide a concise representation of the audio signal suitable for both speech recognition and speaker identification.

- **Fast Fourier Transform (FFT)**: FFT converts audio data from the time domain into the frequency domain, enabling the analysis of spectral content. The algorithm computes a set of sinusoidal components (frequencies and amplitudes) that best describe the signal. By decomposing the signal into its frequency components, FFT provides insights into its harmonic structure and is foundational for many other feature extraction techniques. [1]

- **Log Mel Spectrogram**: A Log Mel Spectrogram visualizes the sound's energy distribution over time and frequency. [10]

  - The signal is first divided into frames and transformed into the frequency domain using FFT.
  - Mel filters are applied to the frequency components to emphasize perceptually significant frequencies.
  - The logarithm of the Mel-filtered energies is taken to better represent human loudness perception.

  Log Mel Spectrograms are often used in deep learning models, as they preserve temporal and spectral information in a visual format.

- **Spectral Centroid**: The spectral centroid represents the "center of mass" of the frequency spectrum. It indicates the perceived brightness of a sound, with higher values corresponding to brighter or sharper tones. The algorithm calculates the weighted mean of frequencies, where weights are their corresponding magnitudes. [15]

- **Chroma Features**: Chroma features represent the distribution of energy across 12 pitch classes (C, C#, D, etc.) in an audio signal. They are particularly useful for analyzing harmonic content and musical signals. The algorithm groups the signal's frequency components into pitch classes, summing their energies to generate a compact representation. [3]

- **Spectral Contrast**: Spectral contrast measures the difference in amplitude between peaks (harmonics) and valleys in the spectrum. This feature is helpful for distinguishing between harmonic and non-harmonic sounds. The algorithm identifies spectral peaks and valleys across multiple frequency bands and computes the amplitude differences. [7]
- **Linear Predictive Coding (LPC)**: LPC models the vocal tract by estimating a set of coefficients that describe the signal's spectral envelope. The algorithm uses autoregressive analysis to predict future signal values based on past samples, effectively encoding the signal while reducing redundancy. LPC is widely used for speech compression and synthesis. [12]
- **Perceptual Linear Prediction (PLP)**: PLP builds on LPC by incorporating psychoacoustic principles, such as the critical-band frequency resolution of human hearing. The algorithm applies a filter bank (similar to Mel filters) to the spectrum, emphasizes important frequency bands, and then estimates the vocal tract shape. PLP provides robust representations of speech by modeling human auditory perception. [6]

2.4 Similarity Learning

Similarity learning is a type of machine learning that focuses on finding patterns of similarity between data points. For audio signals, it helps measure how similar two voice samples are, which is very useful for tasks like voice authentication. The goal is to create a system that brings similar samples closer together and pushes different ones farther apart in a way that the computer can understand.

2.4.1 Application in Voice Authentication

In voice authentication, similarity learning is used to:

- **Verification**: Check if an unknown voice sample matches a specific reference sample to confirm the speaker's identity.

- **Identification**: Compare an unknown voice sample with multiple stored samples in a database to find the closest match.

This approach is helpful because it allows models to compare voice patterns and make accurate decisions, even in cases where voices sound slightly different due to noise or recording conditions.

2.4.2 Loss function in Similarity Learning

Loss functions guide the learning process by teaching the model how to group similar voices together and separate different ones. The most commonly used loss functions in similarity learning are:

- **Contrastive Loss**: This method [4] works with pairs of voice samples and teaches the model to:
  - Bring samples from the same speaker closer in the feature space (a mathematical space where features are represented).
  - Push samples from different speakers farther apart.

  It works by calculating the distance between two samples. If the samples are from the same speaker, the distance is reduced. If they are from different speakers, the distance is increased by a certain margin.

- **Triplet Loss**: This method [14] works with three samples at a time:
  - **Anchor**: The main voice sample.
  - **Positive**: A sample from the same speaker as the anchor.
  - **Negative**: A sample from a different speaker.

  The model is trained to make the anchor closer to the positive sample than to the negative sample, by at least a small margin. This ensures the system learns to group samples correctly for each speaker.

Similarity learning, with the help of these loss functions, is a key method for tasks like voice authentication. It ensures that the system can effectively compare voice samples and make accurate predictions about whether they match or not.

3 Preprocessing

3.1 Gather Addresses

First of all in the preprocessing steps, we store data addresses along with extracted Gender and Student ID's so that we can use these labels in classification problem.

Here we can see the extracted patterns for Gender and Student ID:

```
# Regex patterns for gender
gender_pattern = r"(male|female)"
# Student Id pattern
id_pattern = r"\d{9}"
```

Then store the addresses along with their related labels in a dictionary like this:

```
valid_files[file_path] = {"gender": gender, "ID": file_id}
```

3.2 Denoising

3.2.1 Analyze Signals

This function provides a method for analyzing signal and noise properties in audio files. It is designed to evaluate the signal characteristics of audio files and estimate noise levels by computing the Signal-to-Noise Ratio (SNR). Here is what we do step by step:

- Resample audio files to a uniform sampling rate of 1000 Hz.
- Computes the Root Mean Square (RMS) values of the signal and noise.
- Defines noise as samples below 10% of the maximum amplitude.
- Estimates the SNR using the formula:

$$SNR = 10 \times \log_{10}(\frac{RMS\ of\ signal}{RMS\ of\ noise})$$

```
# Load the resampled audio file
audio, sr = librosa.load(file_path, sr=target_rate)
# Estimate Signal-to-Noise Ratio (SNR)
rms_signal = np.sqrt(np.mean(audio ** 2))
noise_threshold = 0.1 * np.max(audio)
noise_samples = audio[audio < noise_threshold]
rms_noise = np.sqrt(np.mean(noise_samples ** 2)) if len(noise_samples) > 0␣
 ↪else 0
snr = 10 * np.log10(rms_signal / rms_noise) if rms_noise > 0 else float('inf')
```

After going through and analyzing SNR metric for 500 of the audio files, we get to mean SNR (dB) of 1.09 and the plot of SNR versus frequency:

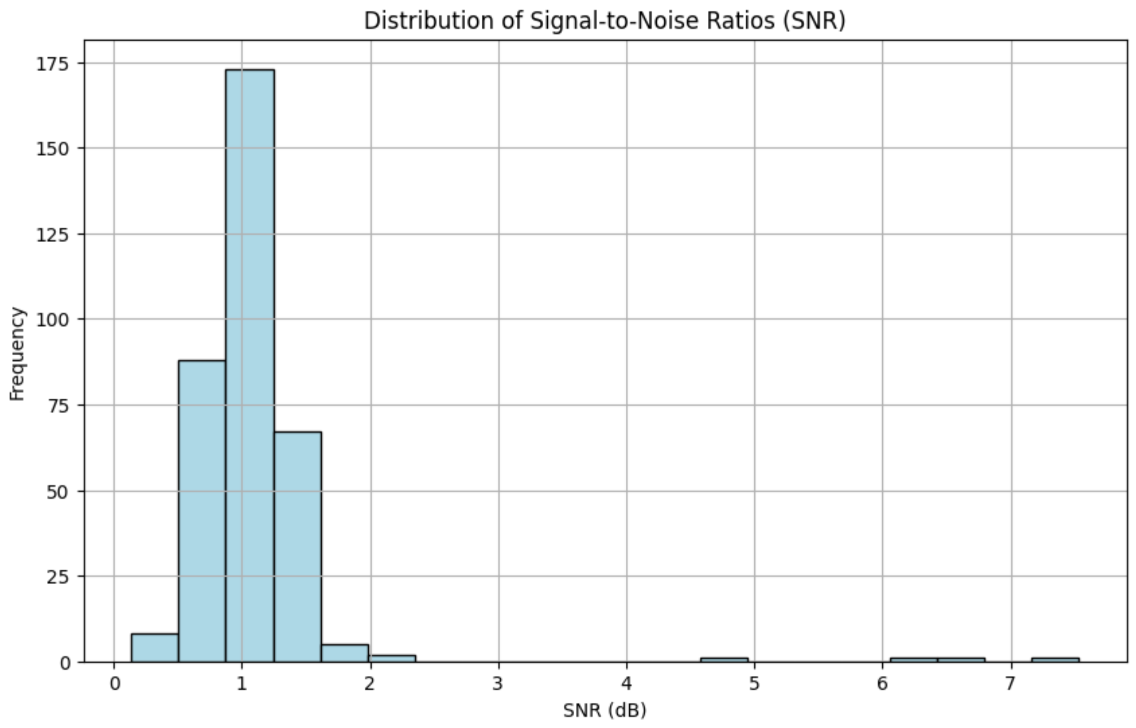

Figure 1: SNR metric versus Frequency of 500 audio files

So we have to apply some noise removing techniques like Bandpass Filtering and Spectrogram Subtraction in order to Denoise the data.

Here we can also see Waveform and Spectrogram of first two audio files:

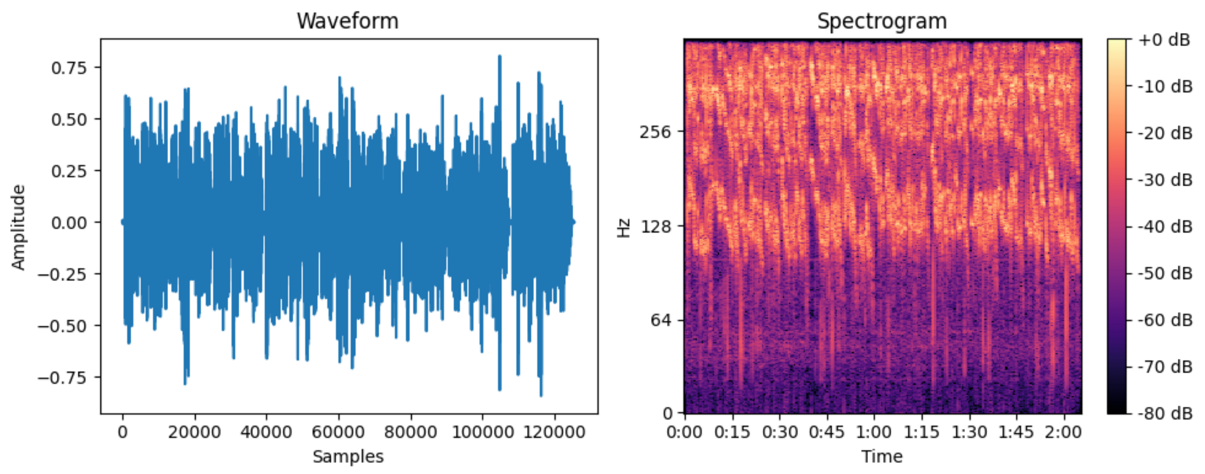

Figure 2: Waveform and Spectrogram of first audio file

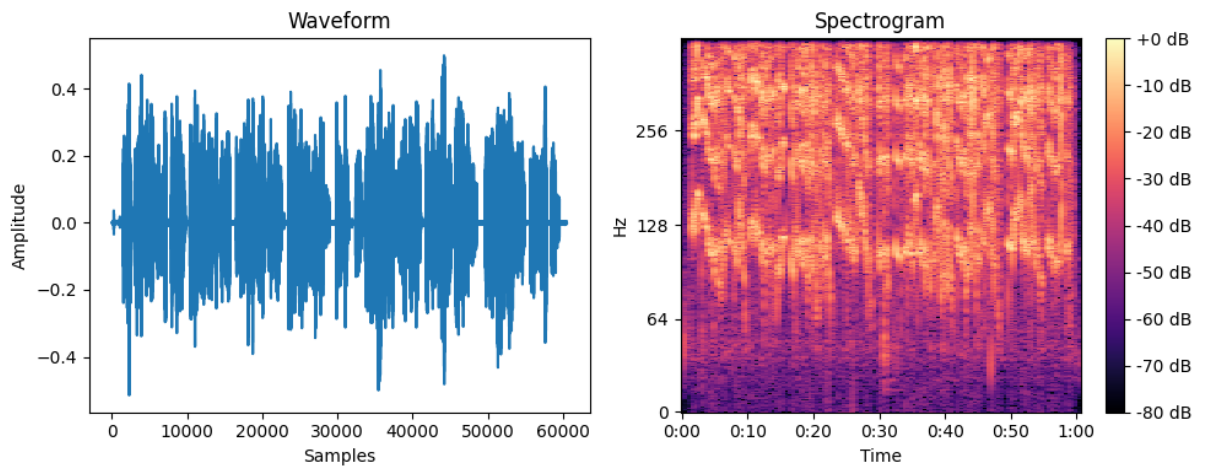

Figure 3: Waveform and Spectrogram of second audio file

Based on these two plots of waveform and spectrogram we can see that the first audio file is longer (2 minutes) and appears more energetic, with higher amplitude variations and a denser waveform. The second file is shorter (1 minute) with slightly lower amplitude, indicating a quieter or more controlled signal. Both spectrograms show rich harmonic content across a wide frequency range, but the first has slightly more intensity in higher frequencies. Vertical patterns in both suggest percussive elements or sudden bursts of energy.

3.2.2 Bandpass Filter

A bandpass filter allows frequencies within a specified range to pass while removing those outside it. This is useful in speech processing, where typical human speech frequencies range from 100 Hz to 8000 Hz, so we can use this filter in order to remove noises that add no value to data with respect to our purpose.

```
def bandpass_filter(audio, lowcut=100.0, highcut=8000.0, fs=1000.0, order=5):
        nyquist = 0.5 * fs
        low = lowcut / nyquist
        high = highcut / nyquist
        b, a = butter(order, [low, high], btype='band')
        return filtfilt(b, a, audio)
```

- **Nyquist Frequency**: Ensures proper filtering by setting the max frequency to half the sampling rate.
- **Butterworth Filter**: Creates a smooth and stable filter response.
- **filtfilt Function**: Applies filtering twice to avoid signal distortion.
- **Use Case**: Helps remove noise and keep only speech-relevant frequencies.

3.2.3 Spectrogram Subtraction

Spectral subtraction is a noise reduction technique that removes background noise by estimating its spectral characteristics and subtracting them from the noisy audio signal. This is especially useful in speech enhancement applications.

```
def spectral_subtraction(audio, sr, noise_estimation_window=1.0, noise_fraction=0.
 ↪2):
        # short-time Fourier transform (STFT)
        D = librosa.stft(audio)
        magnitude, phase = librosa.magphase(D)
        # Estimate the noise
        noise_estimation_samples = int(noise_estimation_window * sr)
        noise_magnitude = np.mean(magnitude[:, :noise_estimation_samples], axis=1)
        # subtract the noise estimate from the magnitude spectrum
        magnitude_denoised = magnitude - noise_fraction * noise_magnitude[:, np.
↪newaxis]
        magnitude_denoised = np.maximum(magnitude_denoised, 0)
        # Reconstruct the signal
        D_denoised = magnitude_denoised * phase
        audio_denoised = librosa.istft(D_denoised)
        return audio_denoised
```

- **STFT** (Short-Time Fourier Transform): Converts the audio signal into the frequency domain.
- **Noise Estimation**: Uses the first part of the signal (assumed to be noise) to estimate background noise.
- **Noise Subtraction**: Reduces the estimated noise from the magnitude spectrum while preserving the phase.
- **Reconstruction**: Uses inverse STFT (ISTFT) to convert the signal back to the time domain.

3.3 Sampling Rate

The sampling rate refers to how many times per second an audio signal is recorded or measured. It is measured in Hertz (Hz).

- A higher sampling rate captures more details of the sound, improving quality.
- A lower sampling rate reduces file size but may lose important sound information.
- The Nyquist theorem [13] states that the sampling rate should be at least twice the highest frequency in the signal to avoid distortion.

3.3.1 Analyze Sampling Rates

Here we can see the sampling rate distribution for audio files. As it is obvious from the distribution, most of the files have a sampling rate of 44100 and 48000 Hz. The unique sampling rates are 44100, 48000, and 24000 and the total mean of about 46200.

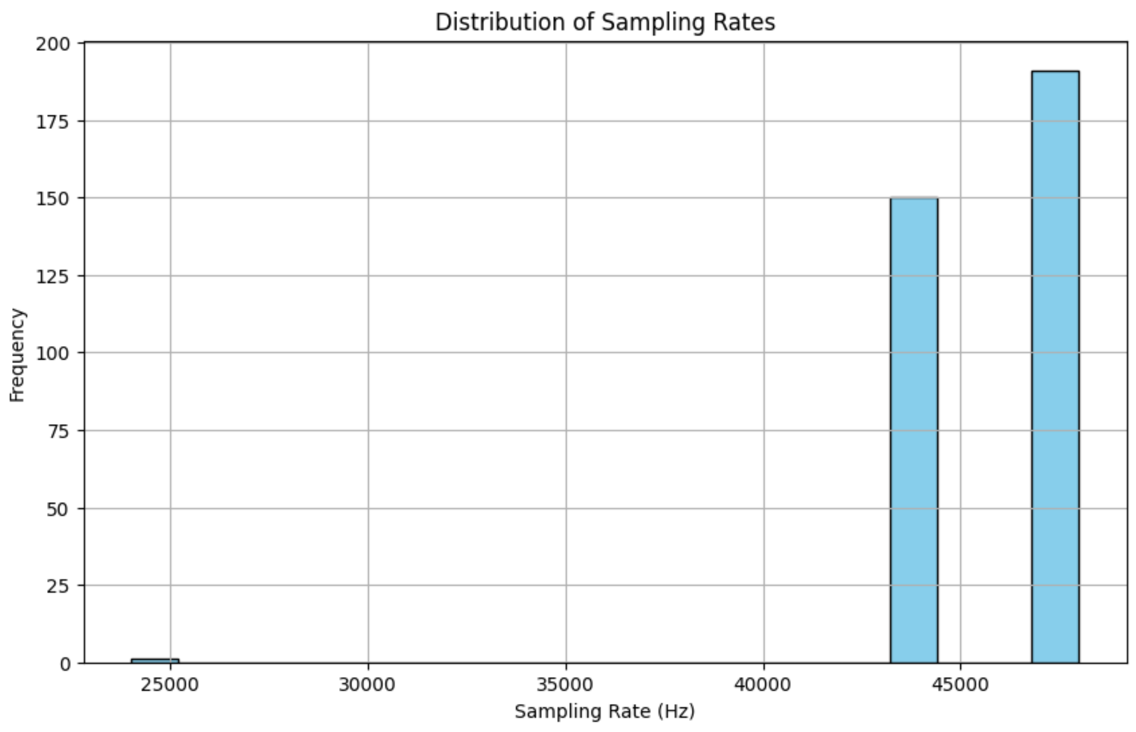

Figure 4: Sampling Rate Distribution for audio files

3.3.2 Resampling

Resampling helps with consistency when combining audio from different sources that may have different sampling rates. Additionally, reducing the sampling rate can speed up processing and save storage, while increasing it can improve quality if needed. As a result, we will resample all the data to 1000 Hz which is a reasonable choice as it is not low and high, but in the middle, preserving complexity, and accuracy with respect to size and computation costs.

```
# Resample to the target rate
resampled_audio = librosa.resample(audio, orig_sr=sr, target_sr=target_rate)
```

3.4 Audio Lengths

We need to have audio samples of equal length to be consistent and accurate in our job. Machine learning algorithms perform well when input features are of equal size, and variations in audio lengths can cause problems with feature extraction and training. Shorter clips might not expose enough speech details, and longer clips might bring in unimportant variations. Normalizing the length enables the model to learn representative patterns equally from all samples.

We can see the Distribution of Audio Lengths in Figure 5. Audio Lengths are not obviously equal, so we have to trim the files in order to get same-length samples.

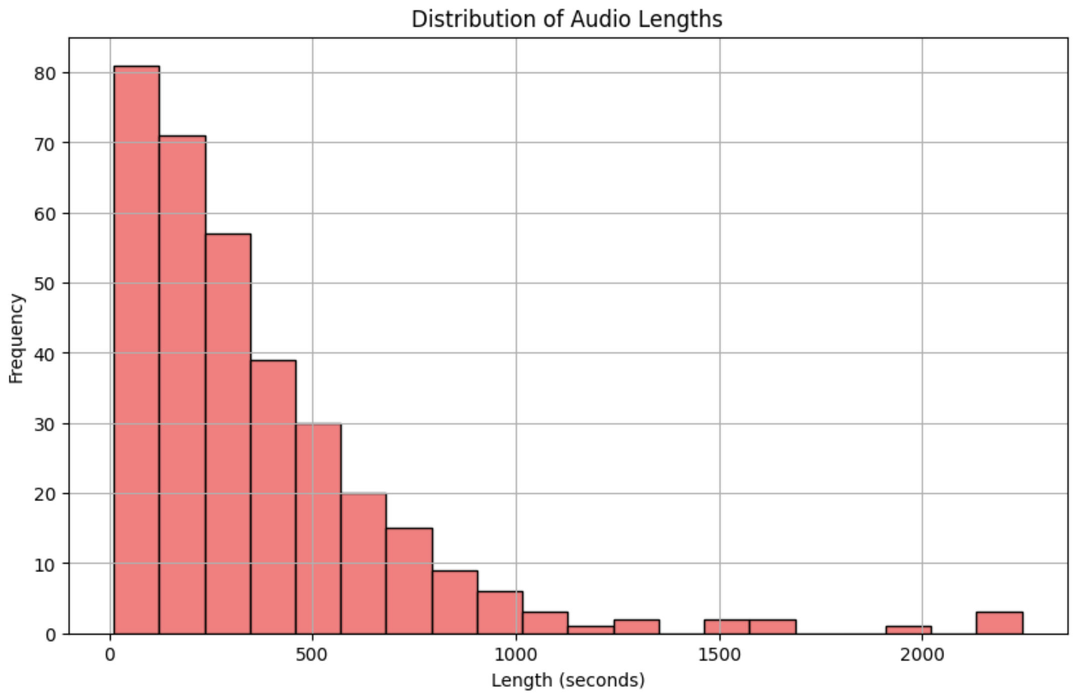

Figure 5: Distribution of Audio Lengths

The mean Audio Length is 360 seconds, but the shortest Audio file is about 10 seconds, so we can trim and split all the files into 10-second samples in order to handle different lengths.

This function normalizes an audio signal so that its values range between -1 and 1 while maintaining the original waveform shape. We normalize the data to ensure consistent volume levels across different recordings. It prevents distortions when processing or feeding the signal into machine learning models. And also helps improve feature extraction and model stability.

```
def normalize_audio(audio):
    return audio / np.max(np.abs(audio)) if np.max(np.abs(audio)) != 0 else audio
```

We split audio files into smaller, fixed-length samples while ensuring consistent sampling and normalization.

- Splits each audio file into 10-second segments.
- Normalizes each segment using defined function.

```
audio, sr = librosa.load(file_path, sr=target_rate)
segment_samples = segment_length * target_rate
num_segments = int(np.floor(len(audio) / segment_samples))
    # Split the audio into smaller chunks
    for i in range(num_segments):
        start_sample = i * segment_samples
        end_sample = (i + 1) * segment_samples
        segment_audio = audio[start_sample:end_sample]
        # Normalize the sample
        segment_audio = normalize_audio(segment_audio)
```

4 Feature Extraction

Until this step, sound data is preprocessed, and each sample is a raw vector with a length. Useful features including MFCC, Log Mel Spectrogram, and energy are extracted from this data and merged.

```
# Load audio file
def feature_extract(audio_path):
  """
  audio_path: path to audio file
  """
  y, sr = librosa.load(audio_path, sr=None)

  # 1. MFCC
  mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)

  # 2. Log Mel Spectrogram
  mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128)
  log_mel_spectrogram = librosa.power_to_db(mel_spectrogram)

  # 3. Energy
  energy = np.sum(y**2) / len(y)

  # Combine features
  combined_features = np.vstack([mfccs, log_mel_spectrogram])
  return combined_features, energy
```

The output's shape is (6147, 2821) (there are 6147 samples which have 2821 features). To reduce the size of feature space to show data and use them for further tasks, its principle components of the convariance matrix are extracted and data is mapped there. To do so, data should be standard.

```
#Extract 3 PCs
scaler = StandardScaler()
df_scaled = scaler.fit_transform(features)
```

Next, to show data in 3D space, three first PCs are extracted.

```
pca = PCA(n_components=3)

PC_features = pca.fit_transform(df_scaled)
```

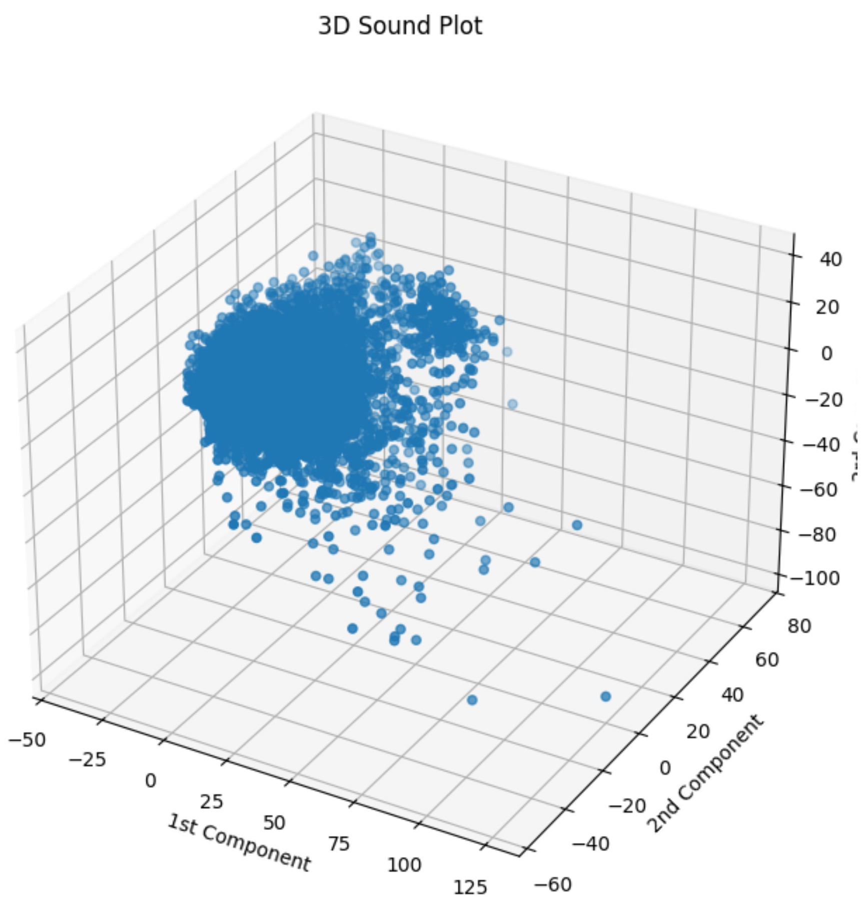

Figure 6: 3D Plot of Data Using 3 First PCs

Moreover, for classification and clustering tasks, first of all, number of PCs which can capture 95% of the variance of data are computed. Then, the standard data is mapped to the space of first PCs of data which can capture this variance.

```
#Proper Number of PComponents

pca = PCA()

pca.fit(df_scaled)

cumulative_variance = np.cumsum(pca.explained_variance_ratio_)

#Best Number of PComponents

n_components_95 = np.argmax(cumulative_variance >= 0.95) + 1

print('Number of Components to Preserve 95% of Variance:', n_components_95)
```

```
Number of Components to Preserve 95% of Variance: 219
```

```
#Transform Data to its PCs
pca_optimal = PCA(n_components=n_components_95)
transformed_data = pca_optimal.fit_transform(df_scaled)
print("Shape of Transformed Data:", transformed_data.shape)
```

```
Shape of Transformed Data: (6147, 219)
```

From now on, data which is used for further tasks is from this shape.

5 Classification

For this task, we will use features extracted from audio files which described in Feature Extraction section. These features have undergone PCA transformation, reducing their dimensionality to 219. Let's first start with Gender Classification. Before getting to gender classification, we have to mention that we have set a seed of 343 so that in different runs, we get the same data and results.

```
np.random.seed(343)
```

We have 113 students' voices in total, so we chose 100 students (50 male and 50 female) for the gender classification task and 10 students (5 male and 5 female) for the Authentication task.

```
unique_ids = data['ID'].unique()
print("There are totally", len(unique_ids), "students")
```

```
There are totally 113 students
```

5.1 Gender Classification

First of all, we prepare the data we have so that we can use it for binary gender classification task. These are the randomly chosen students' IDs:

```
[159403005 159403005 810103106 810101420 810103019 810103106 810600125
 810103022 159403005 151501033 810101575 810102160 159403005 810103317
 159403005 810100130 810103054 159403005 810103146 810103345 159403005
 810199423 810101503 810101575 810600125 810103022 810101465 810102029
 810103019 810103262 810101419 151501033 810600125 810103222 810103008
 810101540 810100130 810199328 810101575 810103106 810101465 810100107
 810103057 810103317 810199569 810101420 810102286 810199423 810101540
 810600125 610300070 810102327 810100135 810199570 810199489 810101456
 810100590 810199489 810103197 810101456 810102027 810199489 810102148
 810101551 810100222 810199489 810100135 810199489 810102027 610300070
 810100135 810101456 810102327 810101456 610300070 810600097 810100268
 810101551 810100075 810199489 810199489 810102017 810102027 810199489
 810100590 810600065 810100193 810100075 810199570 810600088 810102027
 810101551 810101456 810199489 810100135 810100268 810199570 610300070
 610300070 810101551]
```

5.1.1 Data Preperation

These are the number of samples for each gender, so we chose 1300 samples randomly for each gender in order to have an equal number samples for each class.

```
data_gender['Gender'].value_counts()
```

```
1    2298
0    1346
Name: Gender, dtype: int64
```

To use the data for training and testing models, we have to split the data to train and test sets. This is how we do it with regard to have 25% of each class for test set (stratify parameter).

```
X_train_gender, X_test_gender, y_train_gender, y_test_gender =␣
 ↪train_test_split(X_gender, y_gender, test_size=0.25, random_state=42,␣
 ↪stratify=y_gender)
```

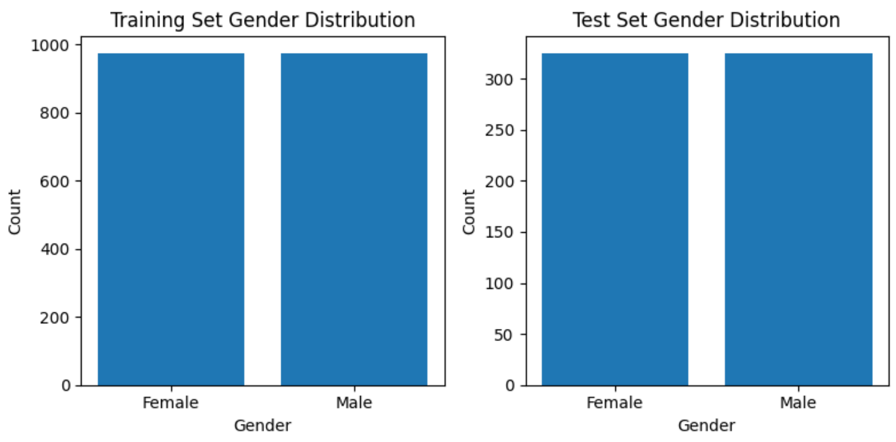

Figure 7: Distribution of samples for each class in train and test sets

So we have a balanced dataset that is ready for training and testing machine learning models.

5.1.2 Normalization

Finally, we normalize the train and test input values using the standardization technique in order to train more robust models.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_gender)
X_test_scaled = scaler.transform(X_test_gender)
```

5.1.3 Model Training

We will train four different models to compare and find out the appropriate model for this with extracted features. These are the models that we are going to work with:

- Logistic Regression
- K-Nearest Neighbor (KNN)
- Support Vector Machine (SVM)
- Multi-Layer Perceptron (MLP)

5.1.4 Logistic Regression

We use sklearn Logistic Regression model for training the data.

```
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

These are some reasons why we chose Logistic Regression as one of the models for gender classification.

- **Baseline Performance**: It is strong baseline model before experimenting with more complex classifiers. If Logistic Regression performs well, more complex models may not be necessary.
- **Low Computational Cost**: It is computationally efficient and quick to train.
- **Works Well with PCA-Reduced Features**: Since PCA-transformed features are often linearly separable, Logistic Regression can effectively classify them without any need for complex decision boundaries.
- **Avoids Overfitting**: Compared to high-capacity models (such as deep neural networks), Logistic Regression has a lower risk of overfitting, particularly with limited training data.
- **Robust to High-Dimensional Data**: Even though 219 features are extracted, Logistic Regression can handle high-dimensional data well.
- **Good for Balanced Datasets**: As the gender classification dataset is balanced, Logistic Regression is likely to perform well without the need for additional balancing techniques.

Here we can see the results of Logistic Regression model for gender classification.

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       325
           1       1.00      0.98      0.99       325

    accuracy                           0.99       650
   macro avg       0.99      0.99      0.99       650
weighted avg       0.99      0.99      0.99       650

Logistic Regression Confusion Matrix:
[[324   1]
 [  5 320]]
```

The Logistic Regression model performed exceptionally well on the gender classification task, achieving **99% accuracy** with high precision, recall, and F1-scores for both classes. The confusion matrix shows only six misclassifications out of 650 samples, so the model is highly reliable. With precision close to 1.00, it rarely makes false positive errors, and its recall of 0.98–1.00 ensures that nearly all actual instances are correctly classified. These results suggest that Logistic Regression effectively captured the patterns in the extracted audio features, making it a strong and efficient choice for this classification task.

5.1.5 KNN

We use sklearn KNN model for training the data.

```
model = KNeighborsClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

These are some reasons why we chose KNN as one of the models for gender classification.

- **Non-Parametric & Flexible**: KNN makes no assumptions about the data distribution, making it suitable for complex and non-linear relationships in audio features.
- **Effective with Small Datasets**: Unlike deep learning models, KNN can perform well with limited training data.

- **Handles Non-Linear Boundaries** – Since gender classification from audio can involve intricate feature distributions, KNN can adapt to non-linear decision boundaries better than linear models like Logistic Regression.

Here we can see the results of KNN model for gender classification.

```
KNN Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.75      0.84       325
           1       0.80      0.97      0.88       325
    accuracy                           0.86       650
   macro avg       0.88      0.86      0.86       650
weighted avg       0.88      0.86      0.86       650

KNN Confusion Matrix:
[[244  81]
 [  9 316]]
```

The KNN model achieved an 86% accuracy in gender classification, showing noticeably lower performance compared to Logistic Regression. The model performed well for Class 1 with 97% recall, meaning it correctly identified most instances of this class. However, Class 0 has a lower recall of 75%, leading to 81 misclassifications, where actual Class 0 samples were wrongly classified as Class 1. This suggests that KNN struggled with distinguishing certain patterns in the extracted audio features, potentially due to overlapping feature distributions.

5.1.6 SVM

We use sklearn SVM model for training the data.

```
model = SVC(probability=True)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

These are some reasons why we chose KNN as one of the models for gender classification.

- **Effective for High-Dimensional Data**: With 219 extracted features, SVM can handle high-dimensional spaces well, making it a strong choice for this task.

- **Good Generalization**: SVM focuses on maximizing the margin between classes, reducing the risk of overfitting.
- **Works Well with Small to Medium-Sized Datasets**: Unlike deep learning models that require large amounts of data, SVM performs well even with limited training samples.

Here we can see the results of SVM model for gender classification.

```
SVM Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       325
           1       0.99      0.98      0.99       325

    accuracy                           0.99       650
   macro avg       0.99      0.99      0.99       650
weighted avg       0.99      0.99      0.99       650

SVM Confusion Matrix:
[[323   2]
 [  6 319]]
```

The SVM model performed well in gender classification, achieving 99% accuracy with high precision, recall, and F1-scores for both classes. The confusion matrix shows only eight misclassifications out of 650 samples, with two false positives and six false negatives, indicating that the model is highly reliable. The balanced precision and recall (both around 0.99) suggest that SVM effectively captured the underlying patterns in the extracted audio features. Overall, SVM proves to be a strong choice, with performance comparable to Logistic Regression but with potentially better handling of complex feature distributions.

5.1.7 MLP

We use Pytorch to implement MLP model for training the data. This is the MLP we have implemented. The outer layer's activation function is sigmoid because we are dealing with binary (0 or 1) classification.

```
class MLP(torch.nn.Module):
    def __init__(self, input_size):
        super(MLP, self).__init__()
        self.layer1 = torch.nn.Linear(input_size, 64)
        self.layer2 = torch.nn.Linear(64, 32)
        self.layer3 = torch.nn.Linear(32, 1)
        self.sigmoid = torch.nn.Sigmoid()

    def forward(self, x):
        x = torch.relu(self.layer1(x))
        x = torch.relu(self.layer2(x))
        x = self.sigmoid(self.layer3(x))
        return x
```

These are some reasons why we chose MLP as one of the models for gender classification.

- **Ability to Learn Complex Patterns**: MLP, with its multiple layers and activation functions, can capture intricate relationships in the 219 extracted features, making it suitable for complex classification tasks.
- **Non-Linearity Handling**: Unlike linear models, MLP uses non-linear activation functions to learn non-linearly separable data distributions, which are common in audio features.
- **Feature Engineering Automation**: MLP can automatically learn feature interactions, reducing the need for extensive manual feature engineering compared to traditional models.
- **Flexibility in Architecture**: The ability to customize the number of hidden layers and neurons allows fine-tuning to optimize performance for gender classification.

We also set the loss function to Binary Cross Entropy loss because of binary classification and the optimizer to Adam.

$$BCE\ Loss = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

```
model = MLP(X_train.shape[1])

criterion = torch.nn.BCELoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Here we can see the results of MLP model for gender classification.

```
Epoch [1/100], Loss: 0.6944
Epoch [11/100], Loss: 0.6341
Epoch [21/100], Loss: 0.5355
Epoch [31/100], Loss: 0.3945
Epoch [41/100], Loss: 0.2433
Epoch [51/100], Loss: 0.1240
Epoch [61/100], Loss: 0.0558
Epoch [71/100], Loss: 0.0261
Epoch [81/100], Loss: 0.0141
Epoch [91/100], Loss: 0.0089

MLP Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.98      0.97       325
           1       0.98      0.95      0.97       325
    accuracy                           0.97       650
   macro avg       0.97      0.97      0.97       650
weighted avg       0.97      0.97      0.97       650

MLP Confusion Matrix:
[[319   6]
 [ 15 310]]
```

The MLP model achieved a 97% accuracy, demonstrating strong performance in gender classification using audio features. The loss consistently decreased over 100 epochs, starting from 0.6944 and reaching 0.0089, indicating successful learning and convergence. The classification report shows high precision (0.96–0.98) and recall (0.95–0.98) for both classes, leading to an overall F1-score of 0.97. The confusion matrix reveals only 21 misclassifications out of 650 samples, with six false positives and 15 false negatives, suggesting that the model is slightly more prone to missing Class 1 instances.

5.1.8 ROC curve analysis

The ROC curve compares the performance of four different models: Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Multi-Layer Perceptron (MLP). The area under the curve (AUC) values indicate their effectiveness in distinguishing between classes.

- Logistic Regression (AUC = 1.00) and SVM (AUC = 1.00) both achieve perfect classification, suggesting they separate the classes perfectly.
- MLP (AUC = 0.99) performs almost as well, with a near-perfect classification ability.
- KNN (AUC = 0.96) shows the lowest performance among the models, though it still performs well with a high AUC score.

Overall, we can conclude from the results that KNN is a bit behind, indicating that it may be more sensitive to the data distribution or requires better hyperparameter tuning.

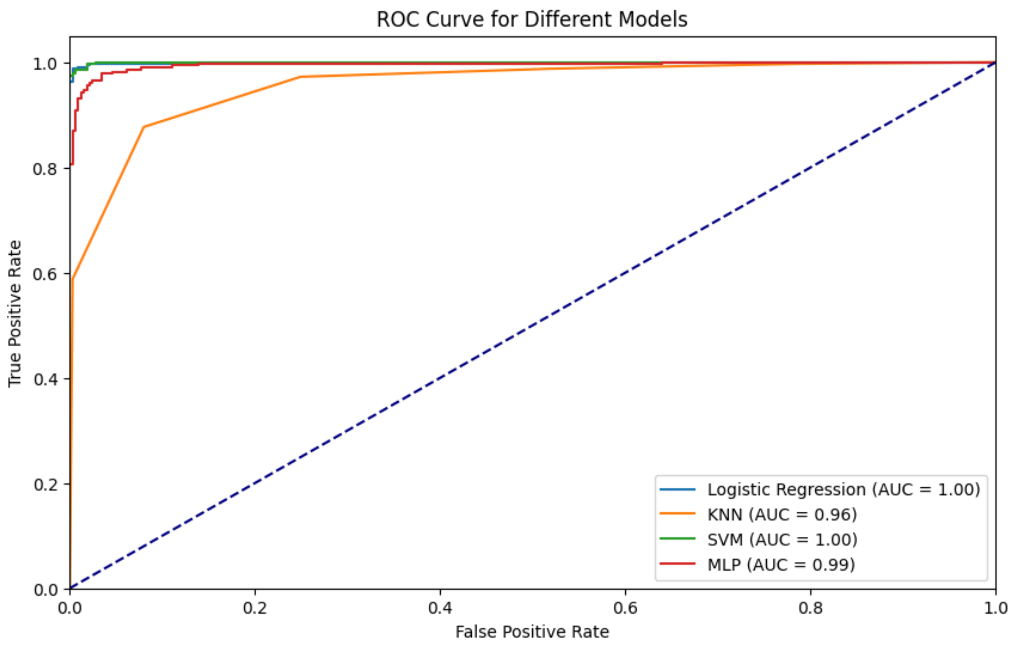

Figure 8: ROC curve for trained models

5.2 Closed-Set Authentication

In this task, we train each model on 10 (5 male and 5 female) students' audio files that have more than 50 samples. This is because with lower samples, we might get error while calculating precision or recall. Then, we chose samples for each student, equal to the number of samples for the least student. Here we can see the distribution of sample numbers for students.

```
[  1,   1,   1,   1,   1,   7,   8,   9,  11,  12,  12,  12,  15,
15,  15,  17,  19,  19,  21,  21,  22,  24,  25,  26,  26,  27,
27,  28,  28,  29,  30,  30,  32,  32,  32,  33,  33,  33,  33,
35,  35,  38,  38,  38,  39,  39,  39,  40,  40,  40,  41,  44,
44,  45,  45,  46,  46,  47,  49,  49,  51,  52,  52,  52,  53,
53,  54,  55,  55,  56,  56,  57,  58,  61,  62,  62,  62,  63,
64,  65,  66,  67,  69,  69,  69,  70,  70,  72,  72,  73,  73,
73,  75,  79,  79,  81,  83,  83,  85,  87,  91,  92, 101, 103,
107, 111, 124, 154, 155, 164, 216, 222, 224]
```

5.2.1 Logistic Regression

Logistic Regression is typically used for binary classification, but it can be extended to multi-class classification using softmax regression.

Here we can see the results of the first run:

```
Selected students are:
[810103295 810101413 810100091 810199417 810103106
 810199489 810199570 810100135 810100168 810101456]


Logistic Regression Classification Report:
              precision    recall  f1-score   support


           0       1.00      1.00      1.00        13
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13
           3       1.00      1.00      1.00        14
           4       1.00      1.00      1.00        14
           5       1.00      1.00      1.00        13
           6       1.00      1.00      1.00        14
           7       1.00      1.00      1.00        13
           8       1.00      1.00      1.00        13
           9       1.00      1.00      1.00        13

    accuracy                           1.00       133
   macro avg       1.00      1.00      1.00       133
weighted avg       1.00      1.00      1.00       133


Logistic Regression Confusion Matrix:
[[13  0  0  0  0  0  0  0  0  0]
 [ 0 13  0  0  0  0  0  0  0  0]
 [ 0  0 13  0  0  0  0  0  0  0]
 [ 0  0  0 14  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0]
 [ 0  0  0  0  0 13  0  0  0  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  0  0  0  0 13  0  0]
 [ 0  0  0  0  0  0  0  0 13  0]
 [ 0  0  0  0  0  0  0  0  0 13]]
```

Here we can see the results of the second run:

```
Selected students are:
[810600125 810102345 810102032 810103054 810103112
 810101551 810600097 810103093 810102148 810101456]

Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13
           3       1.00      1.00      1.00        14
           4       1.00      1.00      1.00        14
           5       1.00      1.00      1.00        13
           6       1.00      1.00      1.00        14
           7       1.00      1.00      1.00        13
           8       1.00      1.00      1.00        13
           9       1.00      1.00      1.00        13

    accuracy                           1.00       133
   macro avg       1.00      1.00      1.00       133
weighted avg       1.00      1.00      1.00       133



Logistic Regression Confusion Matrix:
[[13  0  0  0  0  0  0  0  0  0]
 [ 0 13  0  0  0  0  0  0  0  0]
 [ 0  0 13  0  0  0  0  0  0  0]
 [ 0  0  0 14  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0]
 [ 0  0  0  0  0 13  0  0  0  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  0  0  0  0 13  0  0]
 [ 0  0  0  0  0  0  0  0 13  0]
 [ 0  0  0  0  0  0  0  0  0 13]]
```

Here we can see the results of the third run:

```
Selected students are:
[810600125 810102345 810102032 810103054 810103112
 810101551 810600097 810103093 810102148 810101456]


Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13
           3       1.00      1.00      1.00        14
           4       1.00      1.00      1.00        14
           5       1.00      1.00      1.00        13
           6       1.00      1.00      1.00        14
           7       1.00      1.00      1.00        13
           8       1.00      1.00      1.00        13
           9       1.00      1.00      1.00        13

    accuracy                           1.00       133
   macro avg       1.00      1.00      1.00       133
weighted avg       1.00      1.00      1.00       133
```

```
Logistic Regression Confusion Matrix:
[[13  0  0  0  0  0  0  0  0  0]
 [ 0 13  0  0  0  0  0  0  0  0]
 [ 0  0 13  0  0  0  0  0  0  0]
 [ 0  0  0 14  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0]
 [ 0  0  0  0  0 13  0  0  0  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  0  0  0  0 13  0  0]
 [ 0  0  0  0  0  0  0  0 13  0]
 [ 0  0  0  0  0  0  0  0  0 13]]
```

The Logistic Regression model achieved perfect classification (100% accuracy, precision, recall, and F1-score) across multiple runs on the 10-class authentication task. While this may seem ideal, it raises concerns about potential overfitting. This could indicate that the dataset lacks variability, making classification too easy.

5.2.2 KNN

K-Nearest Neighbors (KNN) can be a strong choice for 10-class authentication due to:

- **Non-Parametric & Flexible**: Makes no assumptions about data distribution and adapts to complex patterns, unlike Logistic Regression.
- **Effective with Structured Features**: Performs well if audio features naturally cluster by class.
- **Handles Non-Linear Boundaries**: Unlike linear models, KNN captures complex decision regions without extra transformations.

Here we can see the results of the first run:

```
Selected students are:
[810103295 810101413 810100091 810199417 810103106
 810199489 810199570 810100135 810100168 810101456]

KNN Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.77      0.87        13
           1       1.00      0.92      0.96        13
           2       1.00      0.15      0.27        13
           3       0.88      1.00      0.93        14
           4       1.00      0.21      0.35        14
           5       1.00      0.15      0.27        13
           6       1.00      1.00      1.00        14
           7       0.22      1.00      0.37        13
           8       0.89      0.62      0.73        13
           9       1.00      0.54      0.70        13

    accuracy                           0.64       133
   macro avg       0.90      0.64      0.64       133
weighted avg       0.90      0.64      0.65       133


KNN Confusion Matrix:
[[10  0  0  0  0  0  0  3  0  0]
 [ 0 12  0  0  0  0  0  1  0  0]
 [ 0  0  2  0  0  0  0 11  0  0]
 [ 0  0  0 14  0  0  0  0  0  0]
 [ 0  0  0  1  3  0  0  9  1  0]
 [ 0  0  0  1  0  2  0 10  0  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  0  0  0  0 13  0  0]
 [ 0  0  0  0  0  0  0  5  8  0]
 [ 0  0  0  0  0  0  0  6  0  7]]
```

Here we can see the results of the second run:

```
Selected students are:
[810600125 810102345 810102032 810103054 810103112
 810101551 810600097 810103093 810102148 810101456]

KNN Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.31      0.47        13
           1       0.43      1.00      0.60        13
           2       0.87      1.00      0.93        13
           3       0.75      0.86      0.80        14
           4       0.61      1.00      0.76        14
           5       1.00      1.00      1.00        13
           6       1.00      0.86      0.92        14
           7       1.00      0.62      0.76        13
           8       0.90      0.69      0.78        13
           9       1.00      0.15      0.27        13

    accuracy                           0.75       133
   macro avg       0.86      0.75      0.73       133
weighted avg       0.85      0.75      0.73       133


KNN Confusion Matrix:
[[ 4  5  0  0  3  0  0  0  1  0]
 [ 0 13  0  0  0  0  0  0  0  0]
 [ 0  0 13  0  0  0  0  0  0  0]
 [ 0  2  0 12  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0]
 [ 0  0  0  0  0 13  0  0  0  0]
 [ 0  1  1  0  0  0 12  0  0  0]
 [ 0  0  1  0  4  0  0  8  0  0]
 [ 0  4  0  0  0  0  0  0  9  0]
 [ 0  5  0  4  2  0  0  0  0  2]]
```

Here we can see the results of the third run:

```
Selected students are:
[810103322 610399205 810101465 810102345 810103269
 810101551 810100135 610300070 810600065 810600097]

KNN Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.64      0.72        14
           1       0.93      1.00      0.96        13
           2       0.81      0.93      0.87        14
           3       0.93      1.00      0.97        14
           4       0.56      1.00      0.72        14
           5       0.88      1.00      0.93        14
           6       1.00      0.86      0.92        14
           7       1.00      1.00      1.00        14
           8       0.62      0.38      0.48        13
           9       1.00      0.50      0.67        14

    accuracy                           0.83       138
   macro avg       0.86      0.83      0.82       138
weighted avg       0.86      0.83      0.82       138


KNN Confusion Matrix:
[[ 9  0  1  0  3  0  0  0  1  0]
 [ 0 13  0  0  0  0  0  0  0  0]
 [ 1  0 13  0  0  0  0  0  0  0]
 [ 0  0  0 14  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0]
 [ 0  0  0  0  0 14  0  0  0  0]
 [ 0  0  0  0  0  2 12  0  0  0]
 [ 0  0  0  0  0  0  0 14  0  0]
 [ 1  1  1  1  4  0  0  0  5  0]
 [ 0  0  1  0  4  0  0  0  2  7]]
```

The three KNN classification runs show variability in accuracy and class-wise performance due to different selected samples. The first run has an accuracy of 64%, with strong precision but poor recall for some classes (e.g., class 2 and 5). The second run improves to 75% accuracy, benefiting from better recall but still struggling with class 9. The third run achieves the best accuracy of 83%, showing more balanced performance across classes, especially for high-recall cases like classes 1, 3, 4, and 7. The inconsistency across runs suggests that KNN's performance is highly sensitive to sample selection, reinforcing the need for robust feature selection and preprocessing to ensure reliable classification.

5.2.3 SVM

Support Vector Machines (SVM) is a good choice for 10-class authentication classification:

- **Effective in High-Dimensional Spaces**: Audio features like MFCCs, spectrograms, and wavelet coefficients create high-dimensional data, which SVM handles well.
- **Robust to Small Sample Sizes**: SVM works well even with limited training samples, making it suitable for speaker authentication where data per user is limited.
- **Generalization**: SVM focuses on maximizing the decision boundary margin, reducing overfitting and improving accuracy.

Here we can see the results of the first run:

```
Selected students are:
[810103295 810101413 810100091 810199417 810103106
 810199489 810199570 810100135 810100168 810101456]


SVM Classification Report:
              precision    recall  f1-score   support


           0       1.00      0.92      0.96        13
           1       1.00      0.92      0.96        13
           2       1.00      1.00      1.00        13
           3       1.00      1.00      1.00        14
           4       0.74      1.00      0.85        14
           5       1.00      0.92      0.96        13
           6       1.00      1.00      1.00        14
           7       1.00      1.00      1.00        13
           8       1.00      1.00      1.00        13
           9       1.00      0.85      0.92        13


    accuracy                           0.96       133
   macro avg       0.97      0.96      0.96       133
weighted avg       0.97      0.96      0.96       133



SVM Confusion Matrix:
[[12  0  0  0  1  0  0  0  0  0]
 [ 0 12  0  0  1  0  0  0  0  0]
 [ 0  0 13  0  0  0  0  0  0  0]
 [ 0  0  0 14  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0]
 [ 0  0  0  0  1 12  0  0  0  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  0  0  0  0 13  0  0]
 [ 0  0  0  0  0  0  0  0 13  0]
 [ 0  0  0  0  2  0  0  0  0 11]]
```

Here we can see the results of the second run:

```
Selected students are:
[810600125 810102345 810102032 810103054 810103112
 810101551 810600097 810103093 810102148 810101456]


SVM Classification Report:
              precision    recall  f1-score   support


           0       0.93      1.00      0.96        13
           1       1.00      1.00      1.00        13
           2       1.00      0.92      0.96        13
           3       1.00      1.00      1.00        14
           4       1.00      1.00      1.00        14
           5       1.00      1.00      1.00        13
           6       1.00      1.00      1.00        14
           7       1.00      1.00      1.00        13
           8       1.00      1.00      1.00        13
           9       1.00      1.00      1.00        13

    accuracy                           0.99       133
   macro avg       0.99      0.99      0.99       133
weighted avg       0.99      0.99      0.99       133


SVM Confusion Matrix:
[[13  0  0  0  0  0  0  0  0  0]
 [ 0 13  0  0  0  0  0  0  0  0]
 [ 1  0 12  0  0  0  0  0  0  0]
 [ 0  0  0 14  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0]
 [ 0  0  0  0  0 13  0  0  0  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  0  0  0  0 13  0  0]
 [ 0  0  0  0  0  0  0  0 13  0]
 [ 0  0  0  0  0  0  0  0  0 13]]
```

Here we can see the results of the third run:

```
Selected students are:
[810103322 610399205 810101465 810102345 810103269
 810101551 810100135 610300070 810600065 810600097]


SVM Classification Report:
              precision    recall  f1-score   support

           0       0.88      1.00      0.93        14
           1       1.00      1.00      1.00        13
           2       1.00      0.93      0.96        14
           3       1.00      1.00      1.00        14
           4       1.00      1.00      1.00        14
           5       1.00      1.00      1.00        14
           6       1.00      1.00      1.00        14
           7       1.00      1.00      1.00        14
           8       1.00      0.92      0.96        13
           9       1.00      1.00      1.00        14

    accuracy                           0.99       138
   macro avg       0.99      0.99      0.99       138
weighted avg       0.99      0.99      0.99       138


SVM Confusion Matrix:
[[14  0  0  0  0  0  0  0  0  0]
 [ 0 13  0  0  0  0  0  0  0  0]
 [ 1  0 13  0  0  0  0  0  0  0]
 [ 0  0  0 14  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0]
 [ 0  0  0  0  0 14  0  0  0  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  0  0  0  0 14  0  0]
 [ 1  0  0  0  0  0  0  0 12  0]
 [ 0  0  0  0  0  0  0  0  0 14]]
```

The results from three different runs of SVM show high performance, with accuracy ranging from 96% to 99%. The model demonstrates strong precision, recall, and F1-scores across all classes, indicating its ability to effectively distinguish between different speakers. However, slight variations in performance across different runs suggest that certain samples have minor misclassifications, as seen in class 4 and class 9 in the first run, where recall dropped slightly. In the second and third runs, the model performed nearly perfectly, with only one or two misclassifications observed in specific classes. This suggests that SVM is reliable for speaker authentication, but the variability across runs may be due to differences in selected samples, indicating that certain speakers or feature distributions are more challenging to classify correctly. Further fine-tuning, feature engineering, or exploring additional kernel functions could help improve robustness across different samples.

5.2.4 MLP

Multilayer Perceptron (MLP) for 10-class authentication classification is a strong choice due to:

- **Feature Learning & Nonlinearity Handling**: MLPs can automatically learn relevant features from representations like MFCCs or spectrograms, capturing complex, nonlinear patterns in audio data that traditional models might miss.
- **Scalability & Computational Efficiency**: MLPs handle large datasets well and use GPUs for faster training, making them more practical than SVMs for high-dimensional audio data.

We used this MLP model for 10-class authentication because it is both simple and has high performance. The hidden layers (64, 32 units) with ReLU help extract meaningful features, while Softmax ensures proper multi-class probability outputs. This lightweight design is efficient and easy to train.

```
class MLP_cls(nn.Module):
    def __init__(self, input_size, num_classes):
        super(MLP_cls, self).__init__()
        self.layer1 = nn.Linear(input_size, 64)
        self.layer2 = nn.Linear(64, 32)
        self.layer3 = nn.Linear(32, num_classes)
        self.softmax = nn.Softmax(dim=1)
    def forward(self, x):
        x = torch.relu(self.layer1(x))
        x = torch.relu(self.layer2(x))
        x = self.layer3(x)
        return self.softmax(x)
```

We used this setup to train the **MLP model** efficiently for **multi-class audio classification**:

- Cross Entropy Loss is used as the loss function because it's the standard choice for multi-class classification.
- Adam is selected as the optimizer for its adaptive learning rate, helping with faster and more stable convergence.

```
model = MLP_cls(X_train.shape[1], y_train.shape[1])
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Here we can see the results of the first run:

```
Selected students are:
[810103295 810101413 810100091 810199417 810103106
 810199489 810199570 810100135 810100168 810101456]

Epoch [1/100], Loss: 2.3025
Epoch [11/100], Loss: 2.2764
Epoch [21/100], Loss: 2.2154
Epoch [31/100], Loss: 2.0988
Epoch [41/100], Loss: 1.9576
Epoch [51/100], Loss: 1.7854
Epoch [61/100], Loss: 1.6341
Epoch [71/100], Loss: 1.5541
Epoch [81/100], Loss: 1.4967
Epoch [91/100], Loss: 1.4709

MLP Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.85      0.88        13
           1       0.93      1.00      0.96        13
           2       0.85      0.85      0.85        13
           3       1.00      1.00      1.00        14
           4       1.00      0.79      0.88        14
           5       0.92      0.92      0.92        13
           6       1.00      1.00      1.00        14
           7       0.93      1.00      0.96        13
           8       1.00      1.00      1.00        13
           9       0.80      0.92      0.86        13
    accuracy                           0.93       133
   macro avg       0.93      0.93      0.93       133
weighted avg       0.94      0.93      0.93       133

MLP Confusion Matrix:
[[11  1  0  0  0  0  0  1  0  0]
```

```
 [ 0 13  0  0  0  0  0  0  0  0]
 [ 0  0 11  0  0  0  0  0  0  2]
 [ 0  0  0 14  0  0  0  0  0  0]
 [ 0  0  1  0 11  1  0  0  0  1]
 [ 1  0  0  0  0 12  0  0  0  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  0  0  0  0 13  0  0]
 [ 0  0  0  0  0  0  0  0 13  0]
 [ 0  0  1  0  0  0  0  0  0 12]]
```

Here we can see the results of the second run:

```
Selected students are:
[810600125 810102345 810102032 810103054 810103112
 810101551 810600097 810103093 810102148 810101456]

Epoch [1/100], Loss: 2.3043
Epoch [11/100], Loss: 2.2828
Epoch [21/100], Loss: 2.2413
Epoch [31/100], Loss: 2.1411
Epoch [41/100], Loss: 1.9793
Epoch [51/100], Loss: 1.8127
Epoch [61/100], Loss: 1.6791
Epoch [71/100], Loss: 1.5864
Epoch [81/100], Loss: 1.5228
Epoch [91/100], Loss: 1.4821

MLP Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.85      0.92        13
           1       1.00      1.00      1.00        13
           2       1.00      0.92      0.96        13
           3       0.93      1.00      0.97        14
           4       0.93      1.00      0.97        14
           5       0.92      0.92      0.92        13
           6       1.00      0.93      0.96        14
           7       0.93      1.00      0.96        13
           8       0.93      1.00      0.96        13
           9       1.00      1.00      1.00        13

    accuracy                           0.96       133
   macro avg       0.96      0.96      0.96       133
weighted avg       0.96      0.96      0.96       133


MLP Confusion Matrix:
[[11  0  0  0  1  0  0  1  0  0]
 [ 0 13  0  0  0  0  0  0  0  0]
 [ 0  0 12  0  0  1  0  0  0  0]
 [ 0  0  0 14  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0]
 [ 0  0  0  1  0 12  0  0  0  0]
 [ 0  0  0  0  0  0 13  0  1  0]
 [ 0  0  0  0  0  0  0 13  0  0]
 [ 0  0  0  0  0  0  0  0 13  0]
 [ 0  0  0  0  0  0  0  0  0 13]]
```

Here we can see the results of the third run:

```
Selected students are:
[810103322 610399205 810101465 810102345 810103269
 810101551 810100135 610300070 810600065 810600097]


Epoch [1/100], Loss: 2.3017
Epoch [11/100], Loss: 2.2759
Epoch [21/100], Loss: 2.2165
Epoch [31/100], Loss: 2.0769
Epoch [41/100], Loss: 1.9203
Epoch [51/100], Loss: 1.7864
Epoch [61/100], Loss: 1.6555
Epoch [71/100], Loss: 1.5341
Epoch [81/100], Loss: 1.4791
Epoch [91/100], Loss: 1.4678
```

```
MLP Classification Report:
              precision    recall  f1-score   support

           0       0.73      0.79      0.76        14
           1       1.00      1.00      1.00        13
           2       0.93      0.93      0.93        14
           3       1.00      1.00      1.00        14
           4       1.00      1.00      1.00        14
           5       1.00      1.00      1.00        14
           6       1.00      1.00      1.00        14
           7       1.00      1.00      1.00        14
           8       1.00      0.77      0.87        13
           9       0.88      1.00      0.93        14
    accuracy                           0.95       138
   macro avg       0.95      0.95      0.95       138
weighted avg       0.95      0.95      0.95       138
MLP Confusion Matrix:
[[11  0  1  0  0  0  0  0  0  2]
 [ 0 13  0  0  0  0  0  0  0  0]
 [ 1  0 13  0  0  0  0  0  0  0]
 [ 0  0  0 14  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0]
 [ 0  0  0  0  0 14  0  0  0  0]
 [ 0  0  0  0  0  0 14  0  0  0]
 [ 0  0  0  0  0  0  0 14  0  0]
 [ 3  0  0  0  0  0  0  0 10  0]
 [ 0  0  0  0  0  0  0  0  0 14]]
```

The results from three different runs of the MLP model for 10-class authentication classification show consistent training performance, with loss values gradually decreasing over epochs. The classification reports indicate high overall accuracy (ranging from 93% to 96%), demonstrating the model's effectiveness. However, performance varies slightly across runs, with some classes achieving perfect recall and precision while others show minor misclassification. The confusion matrices highlight occasional errors, particularly in classes 0 and 8 in different runs, suggesting possible challenges in distinguishing these classes. These variations may stem from differences in selected samples, model initialization, or class similarities. Overall, the MLP demonstrates strong and reliable performance for this task.

5.2.5 Model Comparing

In this 10-class authentication task, each model showed different strengths and weaknesses:

- **Logistic Regression** performed perfectly (100% accuracy) in all runs, which seems unusual. Since logistic regression is a linear model, such high accuracy suggests that the dataset is either highly separable.
- **K-Nearest Neighbors** (KNN) showed the most inconsistency, with accuracy ranging from 64% to 83%. This variation suggests that KNN is highly dependent on the chosen data samples and may not generalize well. Since KNN relies on distance measures, it may struggle with high-dimensional audio data, which can lead to poor performance when features don't clearly separate the classes.
- **Support Vector Machine** (SVM) achieved very strong performance, between 96% and 99% accuracy. SVM is known for handling complex, high-dimensional data well, and it seems to be robust across different data samples. The slight accuracy differences across runs suggest that some samples might be harder to classify, but overall, SVM remains a very reliable choice.
- **Multi-Layer Perceptron** (MLP) had solid accuracy (93% to 96%), but with slightly more Instability than SVM. MLP learns patterns well, but its performance depends on the training process and initialization. It needed more epochs to converge, which means it takes longer to train compared to SVM, but it still showed strong classification ability.

6 Clustering

2 algorithms for clustering are used in this section. Based on figure6, if there is any cluster, it will have concave form. Besides, there is no sign of any cluster with a special shape which can not be captured using algorithms which work based on distance. Therefore, Kmeans and Sequential Clustering are used in this section. Kmeans is useful since it have only one hyperparameter to be specified (K) and when when the shape of clusters are not convex or in a form which cannot be constructed using distance factor. On the other hand, Sequential Clustering is useful because its worst case time complexity is $O(n^2)$, and it has also only one hyperparameter to be specified (cluster distance treshhold), so it can cluster data very fast if the number of clusters is not given.

6.1 Sequential Clustering

In this algorithm, for a new data point, its distance to every cluster is computed. If there is a cluster whose distance is less than a treshhold, the nearest cluster with this property is selected, otherwise a new cluster which contains this new data point is constructed. In each cluster update, centroid of the cluster is also updated and the distance is computed from centroids.

```python
#Sequential Clustering
def sequential_clustering(threshold, data: pd.DataFrame):
    """
    treshhold: treshhold to make a new cluster
    data: data to cluster
    """
    clusters = []
    # centroid of clusters
    means = []
    # predicted labels
    labels = []

    for row in data.values:
        # Ensure it's a NumPy array with float type
        row = np.array(row, dtype=np.float64)
        added = False
        min_dis = float('inf')
        min_i = -1
        #find the nearest cluster if it is less than threshold
        for i in range(len(clusters)):
            distance = np.linalg.norm(row - means[i])
            if distance < threshold and distance < min_dis:
                added = True
                min_dis = distance
                min_i = i
        # check if data is added to a cluster
        if not added:
            labels.append(len(means))
            clusters.append(np.array([row]))
            means.append(row)
        # if it is not added, a new cluster is made
        else:
            # Update mean correctly using the formula:
            # new_mean = (old_mean * cluster_size + new_point) / (cluster_size + 1)
            cluster_size = clusters[min_i].shape[0]
            means[min_i] = (means[min_i] * cluster_size + row) / (cluster_size + 1)

            # Add new row to cluster
            clusters[min_i] = np.vstack([clusters[min_i], row])
            labels.append(min_i)

    return labels
```

It is explored that the minimum distance between 2 points is around 30. Moreover, using distance bigger than 150, data is clustered in one cluster, so in the following code and diagram, distance from 30 to 150 with step of 5 is explored as a treshhold for the sequential learning. For each clustering, silhouette score is computed. The more this score is, the better the clustering is. This score is computed as follows and shows how data within a cluster are compacted and how clusters are far from each other. For a data point i, the silhouette score $s(i)$ is defined as:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{1}$$

where:

- $a(i)$ is the average distance of point i to all other points in the same cluster.
- $b(i)$ is the minimum average distance of point i to all points in the nearest neighboring cluster.

The silhouette score ranges from -1 to 1:

- $s(i) \approx 1$ indicates that the point is well clustered.
- $s(i) \approx 0$ means the point is on the border between clusters.
- $s(i) \approx -1$ suggests that the point is misclassified.

The overall silhouette score for a clustering solution is the mean silhouette score of all points

```
#Explore the Best Parameter for Sequential Clustering
for treshhold in range(30, 150, 5):
  labels = sequential_clustering(treshhold, df_pca.iloc[:, :-2])
  sil_score = silhouette_score(df_pca.iloc[:, :-2], labels)
  scores_seq.append(sil_score)
  if sil_score > max_score:
    best_tresh = treshhold
    max_score = sil_score

print('Best Treshhold for the Sequential Clustering:', best_tresh)
```

```
Best Treshhold for the Sequential Clustering: 145
```

It is obvious in figure9 that almost the more the treshhold is, the more the silhouette score is in this regard. Using the best treshhold (145), data is separated into 2 clusters, as is shown in figure10 (to show this clustering, first 3 PCs are selected).

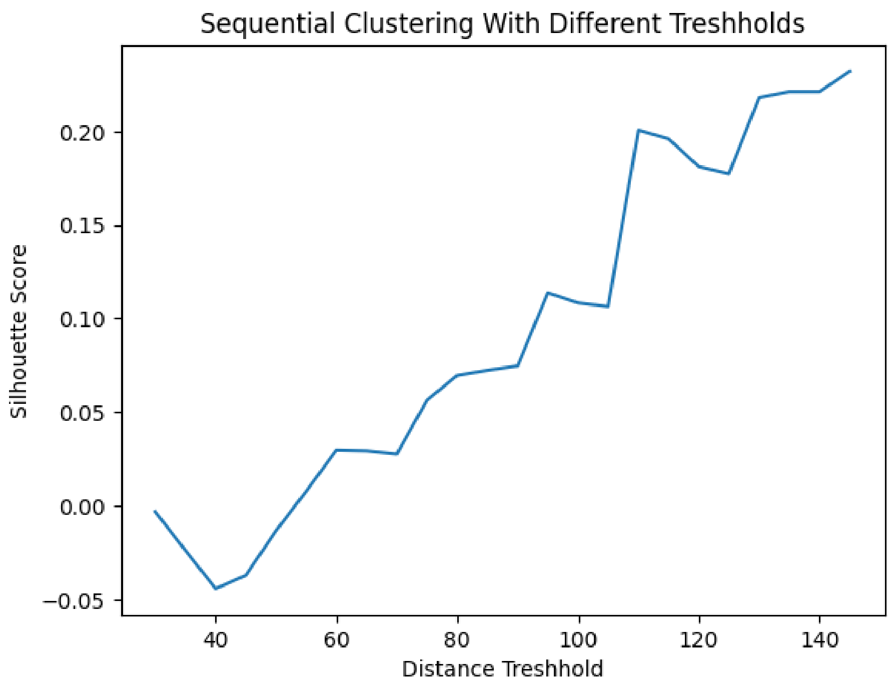

Figure 9: Silhouette/Treshhold Diagram for Sequential Clustering

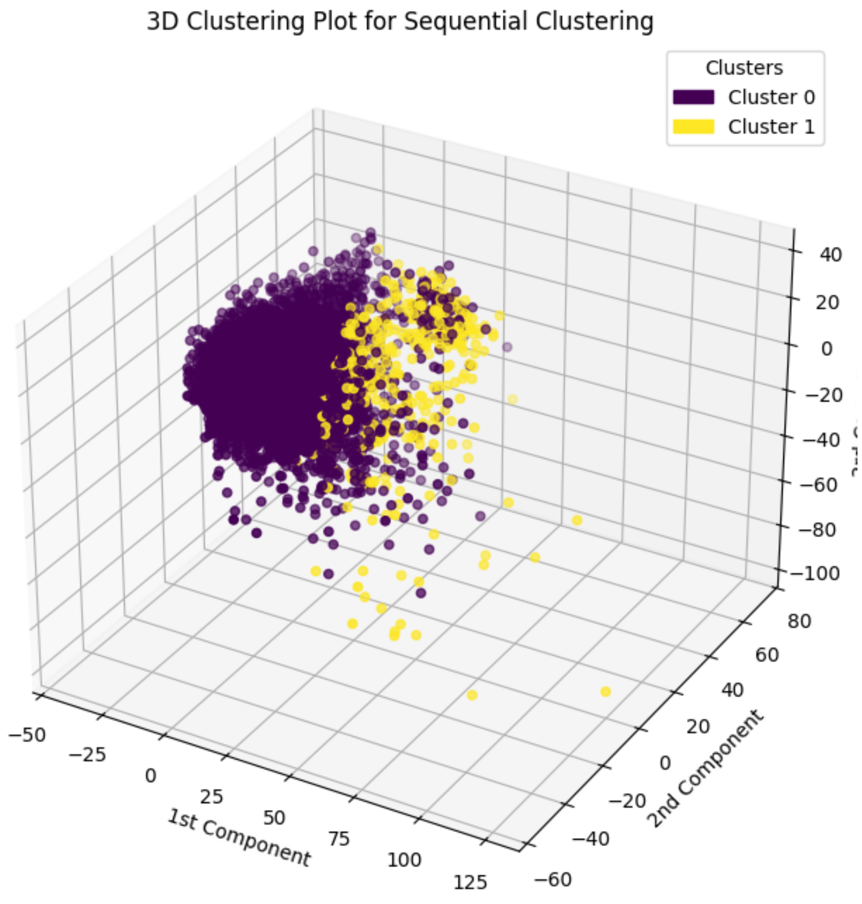

Figure 10: Sequential Clustering Using 145 Treshhold

It can be guessed that data are separated in to 2 groups because the algorithm is trying to separate individuals based on their gender. Therefore, this hypothesis will be explored.

```
# Exlore Seq_Clustering
percentages = (
    pca_clusters.groupby(['Gender', 'seq'])['ID']
    .count()
    .groupby(level=0)  # Normalize within each 'Gender' group
    .apply(lambda x: 100 * x / x.sum())
)

display(percentages)
```

```
Gender  Cluster Percent
0       0       86.213469
        1       13.786531
1       0       96.960420
        1        3.039580
Distribution of Clusters within Genders in Sequential Clustering
```

In this table, 0 is for female and 1 is for male gender. The seq column is sequential clustering label of each data. The last column is also distribution of clusters in each gender. Cluster 0, Therefore, encompass about 86% of males and about 97% females. As a result, this separation is not based on the gender of individuals. Moreover, It was also obvious in the figure10 that data is not distributed within clusters equally (yellow cluster is thinner). It maybe because of existing outliers in the dataset as is also showed in the figure. Outliers may draw the centroid of the yellow cluster to themselves, and the data which is far from the overall centroid of the dataset is considered as a separate cluster, and the other data is also considered as the purple cluster.
To make sure that the clustering is implemented correctly, we make sure that all samples of each individual are considered in one of the clustered.

```
dist = (
    pca_clusters.groupby(['ID'])['seq']
    .apply(lambda x: max(100 * x.sum() / len(x), 100 * (1-(x.sum()) / len(x))))
)
display(dist)
```

```
ID           Percent
151501033    100.000000
159403005     99.553571
610300017    100.000000
610300032    100.000000
610300070    100.000000
                ...
810600133     88.888889
810800024     75.000000
810801072    100.000000
810801075    100.000000
810899073     93.333333
Each individual's Distribution Clusters (Maximum Cluster Percent is Shown)
```

```
counter = (
    pca_clusters.groupby(['ID'])['seq']
    .count()
)
print('Average Percent of Existance of Each Individual in Each Class:',(counter *␣
 ↪dist).sum() / counter.sum())
```

```
Average Percent of Each Individual in Its Maximum Cluster: 95.23
```

In the code above, we explored how each individual is distributed in clusters. Besides, We explored average percent of individuals' clustering in their maximum cluster which is around 95%, so the algorithm is correctly separating sounds based on their properties (which is not related to their gender). Yellow cluster encompasses data which is outlier (or near outliers) and the purple cluster encompasses data which considered are nearer to core of sounds which have average (more relevant) sound properties. Further, we explore that if most of samples of each person are considered in the purple class and the yellow class (outliers) are their samples which have noise (or other properties that make them distinct from their other sounds).

```
counter = (
    pca_clusters.groupby(['ID'])['seq']
    .apply(lambda x: (1-x).sum() / len(x))
)
```

```
print("People whose voices' maximium cluster is not 0:", counter[counter < 0.5].
 ↪index)
```

```
IDs whose voices' maximium cluster is not 0: 810199489
```

Therefore, the guess is not always true since maximum cluster for ID 810199489 is 1 (the yellow cluster). However, it can be weakly shown that the majority of the sounds of each individuals are recorded in a same (or maybe proper) condition, while some of the recordings of some of the individual are recorded in a different condition (maybe a bad condition). The first group is the purple cluster and the second one is the yellow cluster.

Using this algorithm, we also try to cluster individuals based on their IDs, so firstly, we find the distance which leads to a number of clusters which is equal to the number of individuals, then we try to cluster data using sequential algorithm and check if individuals are divided in to separate clusters.

```
print('Number of Individuals:', df_pca.ID.unique().shape[0])
```

```
Number of Individuals: 113
```

We have explored relation of distance and the number of clusters, and distance 70 clusters data results to about 113 groups (number of individuals).

```
#Explore Seq_Cluster with Other Treshholds (Classes Equal to Number of Students)
count_labels = []
#Explore number of clusters corresponding to number distance treshhold
for i in range(30, 100, 5):
  labels = sequential_clustering(i, df_pca.iloc[:, :-2])
  count_labels.append(len(np.unique(labels)))
```

Therefore, based on this treshhold, a sequential clustering algorithm is implemented and the distribution of individuals is explored.

```
dist_best = 5 * (np.argmin(np.abs(np.array(count_labels) - df_pca.ID.unique().
 ↪shape[0]))) + 30
labels = sequential_clustering(dist_best, df_pca.iloc[:, :-2])
print("Distance which Leads to 113 Number of Clusters:", dist_best)
```

```
Distance which Leads to 113 Number of Clusters: 70
```

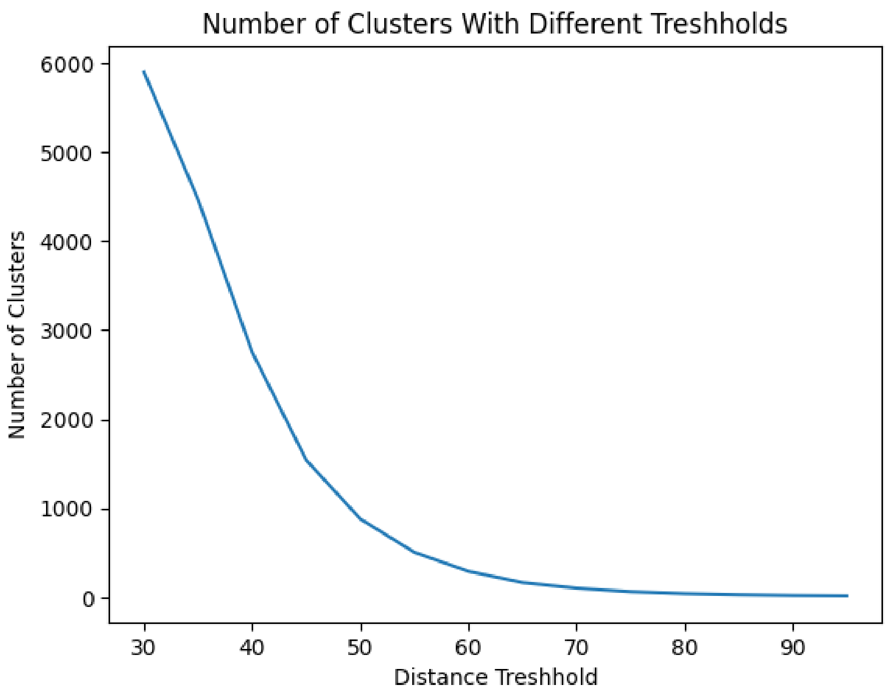

Figure 11: Number of Clusters/Treshhold for Sequential Clustering

```
pca_clusters['seq'] = labels_seq
dist = (
    pca_clusters.groupby(['ID', 'seq'])['Gender']
    .count()
)
display(dist)
```

```
ID         Cluster Count
151501033  0       48
           1        1
           2        1
           3        1
           4        1
                   ..
810801075  38       2
810899073  0        9
           2        1
           37       4
           122      1
```

The table result, shows cluster distribution of IDs. It shows that individuals are separated into clusters not based on their IDs. It means that this number of clusters with distance 70 is not related to number of individuals. Based on the figure11, the smaller numbers result in smaller number of clusters and the bigger distances treshhold result in bigger number of clusters.

6.2 Kmeans

In this algorithm, first of all, k number of random points are selected as centroids of k clusters. Next, in each iteration, points are assigned to the nearest cluster and centroids are updated accordingly. This process continues until no centroids are converged.

In this regard, in the first step, the best number of clusters (k) is selected based on silhouette score. The process of selection is like the process of selection of distance treshhold in the sequential clustering algorithm.

```
#Explore the Best Parameter for Kmeans
max_score = 0
best_k = 0
scores_kmean = []
for k in range(2, 116, 5):
  labels = k_means(k, df_pca.iloc[:, :-2])
  sil_score = silhouette_score(df_pca.iloc[:, :-2], labels)
  scores_kmean.append(sil_score)
  if sil_score > max_score:
    best_k = k
    max_score = sil_score

print('Best K for the Kmeans:', best_k)
```

```
Best K for the Kmeans: 2
```

We tried Ks between 2 and 113 (number of individuals) with step 5. The best one is 2. Again, a strong guess is that this number of clusters is because of the factor of gender. It is explored as follows.

```
labels_kmeans = k_means(best_k, df_pca.iloc[:, :-2])
```

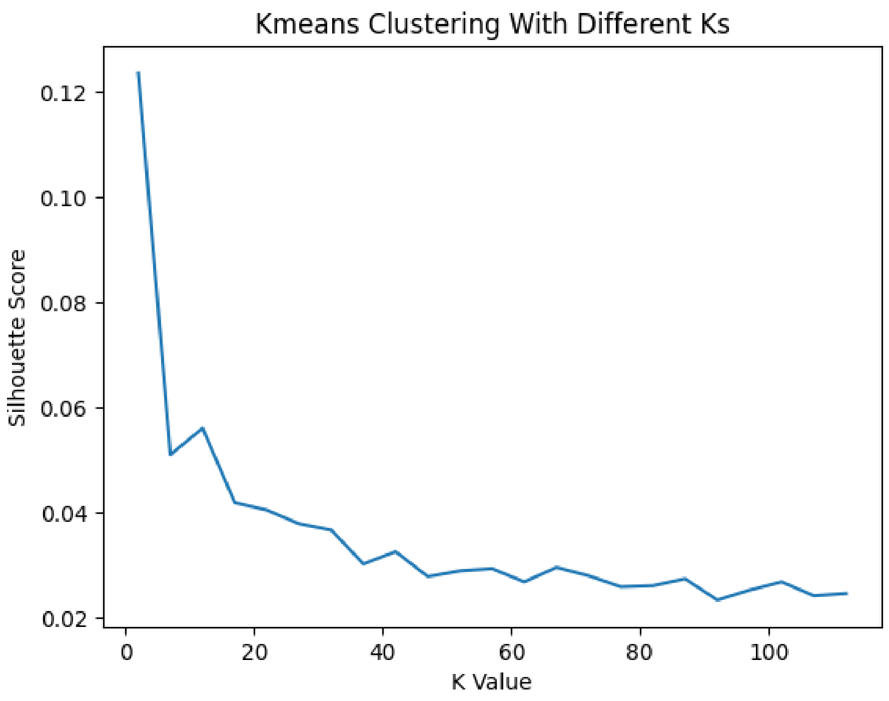

Figure 12: Kmeans Clustering with Different Ks

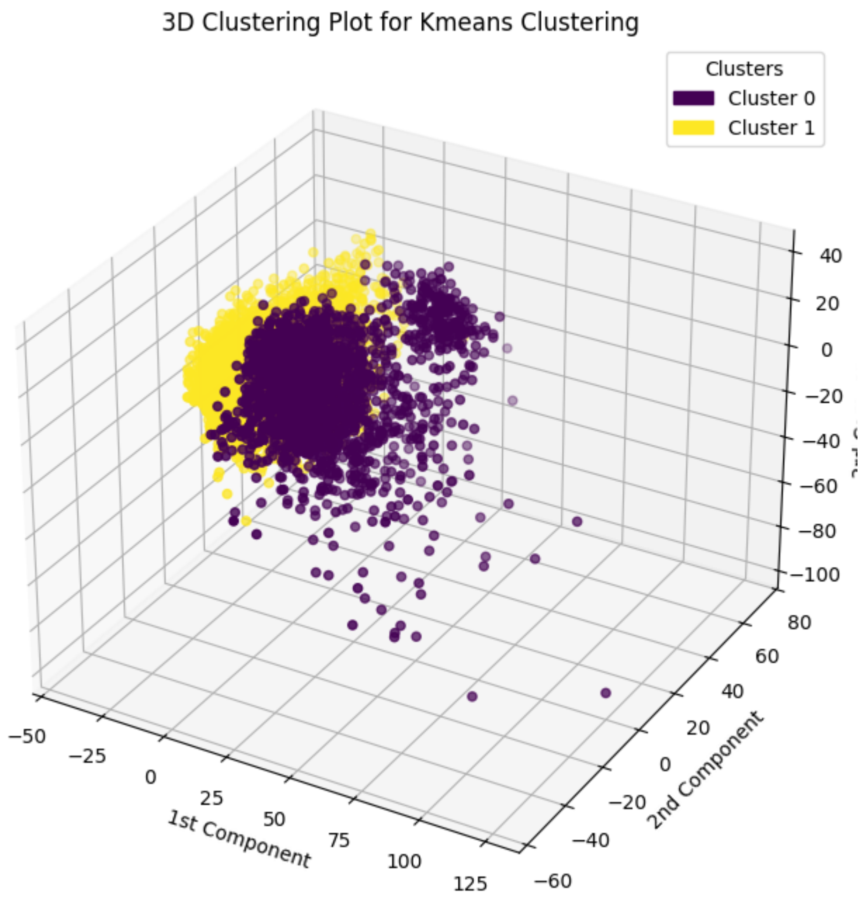

Figure 13: Kmeans Figure (3 First PCs are Selected)

```
#Explore Kmeans
pca_clusters = df_pca.loc[:, ['ID', 'Gender']]
pca_clusters['kmeans'] = labels_kmeans
pca_clusters['seq'] = labels_seq
percentages = (
    pca_clusters.groupby(['Gender', 'kmeans'])['ID']
    .count()
    .groupby(level=0)  # Normalize within each 'Gender' group
    .apply(lambda x: 100 * x / x.sum())
)

display(percentages)
```

```
Gender  Cluster    Percent
0       0         94.282084
        1          5.717916
1       0         13.535972
        1         86.464028
```

As mentioned, Males are 1 and Females are 0 in our dataset. The table shows that Males are mostly in cluster 1 and Females are mostly in cluster 0. Therefore, our guess works out and the Kmeans algorithm has clustered data based on their gender. This shows that these 2 genders are separated using an clustering algorithm which works based on distance. Therefore, sequential algorithm could also do so. The reason of lack of success for sequential learning algorithm might have been because of order of the samples; This algorithm is sensitive to the order. Dataset is ordered based on IDs for each of these algorithms, so genders are not seen in a order; This may be the reason.

Like sequential algorithm, Kmeans algorithm is also explored with number of clusters equal to number of individuals (113). Next, we explore if individuals are separated in to distinct clusters.

```
pca_clusters['kmeans'] = labels_kmeans
dist = (
    pca_clusters.groupby(['ID', 'kmeans'])['Gender']
    .count()
)
display(dist)
```

```
ID          Cluster    Count
151501033   10         2
            12         2
            15         1
            18         1
            21         1
                      ..
810899073   48         1
            85         1
            87         1
            101        2
            107        1
```

Due to the table, the hypothesis is rejected and using Kmeans, data cannot be clustered into 113 each containing each individual's samples. There can be many reasons such as lack of being separatablity of individuals' samples, effect of initial point in Kmeans and so on. As mentioned, data is ordered based on IDs, while sequential clustering algorithm was still unable to cluster individuals into separate groups. Therefore, samples of each individual is more likely to be unseparateable based on distance factor. This fact will be explored. To explore, clusters 21 and 26 are taken. From cluster 21 and 26, 2 vectors for ID 151501033 and from cluster 26, one vector for ID 810100130 is selected. As it is showed in the following code, distance between the sample of 810100130 and the sample of 151501033 from cluster 26 is less than distance between the 2 samples of 151501033.

```python
new_pca = df_pca
new_pca['kmeans'] = pca_clusters['kmeans']
id_1_group_1 = new_pca[(new_pca['kmeans'] == 21)].iloc[0, :-3]
id_2_group_2 = new_pca[(new_pca['kmeans'] == 21)].iloc[1, :-3]
id_1_group_2 = new_pca[(new_pca['kmeans'] == 26)].iloc[0, :-3]
print('First Vector: ID', new_pca[(new_pca['kmeans'] == 21)].iloc[0, -3], 'Sample␣
 ↪From Cluster', new_pca[(new_pca['kmeans'] == 21)].iloc[0, -1])
print('Second Vector: ID', new_pca[(new_pca['kmeans'] == 21)].iloc[1, -3], 'Sample␣
 ↪From Cluster', new_pca[(new_pca['kmeans'] == 21)].iloc[1, -1])
print('Third Vector: ID', new_pca[(new_pca['kmeans'] == 26)].iloc[0, -3], 'Sample␣
 ↪From Cluster', new_pca[(new_pca['kmeans'] == 26)].iloc[0, -1])
```

```
print('Distance of Vector One and Three:', np.linalg.norm(id_1_group_1 -␣
 ↪id_1_group_2))
print('Distance of Vector Three and Two:', np.linalg.norm(id_1_group_2 -␣
 ↪id_2_group_2))
```

```
First Vector: ID 151501033 Sample From Cluster 21
Second Vector: ID 810100130 Sample From Cluster 21
Third Vector: ID 151501033 Sample From Cluster 26
Distance of Vector One and Three: 66.84590220095198
Distance of Vector Three and Two: 66.55363880373801
```

Therefore, samples from one individual are not separateable using the distance factor. However, these 2 distances are very near each other. This fact, along with accuracy of classification tasks, show quality of extracted features. Nevertheless, using more features, lack of using PCA (or using more PCs), and etc. may leverage quality of clustering.

7 References

References

[1] E. Oran Brigham and Roy E. Morrow. The fast fourier transform. *IEEE Spectrum*, 4(12):63–70, 1967.

[2] Steven Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366, 1980.

[3] T. Fujishima. Real-time chord recognition of musical sound: A system using common lisp music. In *Proceedings of ICMC*, pages 464–467, 1999.

[4] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Proceedings of CVPR*, pages 1735–1742, 2006.

[5] Fredric J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[6] H. Hermansky. Perceptual linear predictive (plp) analysis of speech. *The Journal of the Acoustical Society of America*, 87(4):1738–1752, 1990.

[7] D. N. Jiang, L. Lu, H. J. Zhang, J. H. Tao, and L. H. Cai. Music type classification by spectral contrast feature. In *Proceedings of ICME*, pages 113–116, 2002.

[8] Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. Audio augmentation for speech recognition. In *Proceedings of Interspeech*, pages 3586–3589, 2015.

[9] Arun Kumar and Ashok Sriram. An analysis of pre-processing techniques for speech recognition. *Journal of Signal and Information Processing*, 10(1):34–43, 2019.

[10] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *Proceedings of ISMIR*, pages 93–96, 2000.

[11] Philipos C. Loizou. *Speech enhancement: Theory and practice.* CRC Press, 2013.

[12] J. D. Markel and A. H. Gray. *Linear Prediction of Speech.* Springer Science & Business Media, 1976.

[13] Harry Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47(4):617–644, 1928.

[14] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of CVPR*, pages 815–823, 2015.

[15] George Tzanetakis and Perry Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, 2002.

[16] Junichi Yamagishi, Christophe Veaux, Simon King, and Steve Renals. Speech synthesis technologies for individuals with vocal disabilities: Voice banking and reconstruction. *Acoustical Science and Technology*, 33(1):1–5, 2012.