Voice Biometrics

Unlocking Identity Through the Power of Speech

From Gheffari Nour El Houda

M1 Data Science and NLP

university of Saad Dahleb Blida

2024/2025

Introduction

Voice is the most natural way that humans have to communicate with each other. As a signal, the voice is quite complex, conveying multiple types of information, including, of course, information about the identity of the speaker. Discovering the identity of the speaker is the ultimate goal of the science of voice biometrics, which is also the goal of this research.

The aim of this project is to explore voice biometrics, with a focus on speaker verification. We seek to build a system capable of identifying a speaker by analyzing the unique characteristics of their voice.

Voice biometrics is a fascinating field that combines signal processing, machine learning, and linguistics. This project was chosen due to its increasing importance in practical applications such as security, authentication, and voice assistants. Additionally, it provides a unique opportunity to apply theoretical concepts in automatic speech processing (ASR) and data science.

In this project, we aimed to build a speaker verification system using statistical methods, particularly i-vectors and Gaussian Mixture Models (GMMs). We also explored acoustic feature extraction techniques, such as Mel-Frequency Cepstral Coefficients (MFCCs), to capture relevant voice information.

History

Definition and Approaches

Voice Biometrics: Definition and Applications

Voice biometrics is a technology that identifies or verifies individuals based on their unique vocal characteristics. It is widely used in security systems, banking authentication, access control, and customer service automation. The uniqueness of a person's voice arises from physiological and behavioral traits, making it a viable biometric identifier.

Early Approaches (1950s - 1980s): Acoustic Parameter-Based Methods

The earliest voice biometrics methods relied on acoustic parameters such as fundamental frequency (F0), pitch, timbre, and formants. These features were manually extracted and compared to stored voice samples. While these methods were simple and interpretable, they suffered from high variability due to changes in emotion, health, and environmental noise. Consequently, they were not reliable for large-scale or high-security applications.

Statistical and Machine Learning Approaches (1990s - 2010s): I-Vectors and GMM-UBM

With advances in statistical modeling, voice biometrics shifted towards probabilistic approaches like Gaussian Mixture Models - Universal Background Model (GMM-UBM) and later, the i-vector framework. The i-vector approach, introduced in the late 2000s, transformed high-dimensional voice features into a lower-dimensional representation, allowing efficient speaker verification. Machine learning techniques such as Support Vector Machines (SVMs) and Hidden Markov Models (HMMs) further improved accuracy. These methods significantly enhanced performance compared to acoustic-based techniques and became the standard for many voice authentication systems.

Deep Learning Approaches (2010s - Present): Neural Networks and End-to-End Models

With the rise of deep learning, neural networks have revolutionized voice biometrics. Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformer-based models now learn hierarchical representations of voice data, reducing the need for manual feature engineering. End-to-end speaker recognition models, such as x-vectors, outperform traditional approaches by leveraging large-scale datasets and optimizing feature extraction and classification in a single framework. While deep learning provides state-of-the-art accuracy, it requires significant computational resources and large amounts of labeled data for training.

Our Approach for the project, we used statistical and machine learning methods, specifically the i-vector framework, due to its balance between accuracy and efficiency. This approach allows us to achieve reliable speaker verification while maintaining interpretability and computational feasibility.

Definition and Theorical Background

Spectral Analysis

Spectral analysis involves examining how the energy of a speech signal is distributed across different frequencies. Since every speaker has unique vocal tract characteristics, analyzing the frequency components of speech allows us to extract speaker-specific features.

Spectral Envelope

The spectral envelope represents the smooth curve that outlines the peaks of a speech signal's frequency spectrum. It captures the resonant frequencies (formants) of the vocal tract and plays a crucial role in speaker recognition. By analyzing the spectral envelope, we can extract features that characterize a speaker's unique vocal properties while minimizing the influence of pitch and background noise.

Mel-Frequency Cepstral Coefficients (MFCCs)

MFCCs are one of the most widely used features in speech processing. They simulate the human auditory system by:

1/Applying a Fourier Transform to the speech signal.

2/Passing the spectrum through a Mel-scale filter bank, which mimics human hearing sensitivity.

3/Taking the logarithm of the filter bank outputs to handle amplitude variations.

4/Computing the Discrete Cosine Transform (DCT) to obtain decorrelated coefficients.

5/MFCCs are effective because they capture the vocal tract shape, which is a distinguishing factor for speaker recognition.

Gaussian Mixture Model (GMM)

A GMM is a probabilistic model that represents a speaker's voice characteristics using a mixture of Gaussian distributions. Given a speech sample, GMM models the probability distribution of feature vectors (e.g., MFCCs) by estimating the likelihood that they belong to a specific speaker. The speaker with the highest likelihood score is identified as the speaker of the given voice sample.

Universal Background Model (UBM)

A UBM is a large, speaker-independent GMM trained on a diverse dataset of speakers. Instead of training a separate model for each speaker from scratch, speaker-specific GMMs are adapted from the UBM using Maximum A Posteriori (MAP) adaptation. This helps in improving recognition accuracy, especially when limited training data is available for individual speakers.

Baum-Welch Statistics

Baum-Welch statistics are used in speaker verification systems to extract sufficient statistics from speech data when using GMM-UBM models. They include:

1/Zero-order statistics: The number of times a given Gaussian component is activated for a speech sample.

2/First-order statistics: The sum of feature vectors weighted by their probability of belonging to a particular Gaussian component.
These statistics help in estimating speaker models more effectively, particularly in the i-vector framework.

I-Vectors (Identity Vectors)

The i-vector (identity vector) approach is a powerful feature extraction technique for speaker recognition. It represents a speech utterance as a single low-dimensional vector, capturing speaker-specific traits.I-vectors improve upon GMM-UBM by consolidating all speaker characteristics into a compact representation, making speaker recognition more efficient.

T-Matrix (Total Variability Matrix)

The T-matrix is used in the i-vector model to transform speech features (such as GMM-UBM statistics) into a lower-dimensional space. It captures both speaker-dependent and channel-dependent variations, allowing the system to model the unique voice

characteristics while reducing redundancy. The T-matrix is trained using large amounts of data to generalize well across different speakers.

Probabilistic Linear Discriminant Analysis (PLDA)

Probabilistic Linear Discriminant Analysis (PLDA) is a statistical technique widely used in speaker verification to improve the discrimination between speakers when using i-vectors or x-vectors. It models the variability of speech data in a way that enhances speaker separability.

Methodology

Steps and code explanation

The main workflow we followed was to first extract MFCCs from the voice data. Then, we trained the UBM (Universal Background Model) on these extracted features to identify hidden patterns in human speech. After this, we calculated the Baum-Welch statistics and used a PCA model to create and train the T-matrix, which we stored for later use. Once we had the generalized UBM model and the T-matrix, we could generate user-specific UBMs and compute i-vectors, which were later compared for speaker recognition.

This workflow was divided into two main phases: the training phase and the testing phase.

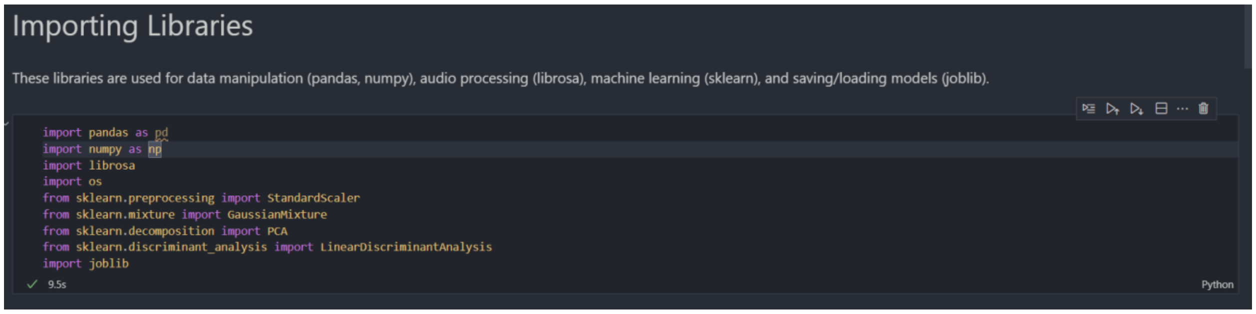
Training phase :

In this phase our main objective was to train the UBM model so that it could learn patterns in human voice data. To do this, we used the Common Voice dataset, which contains various voice clips from random speakers. Once the UBM was trained, we saved it using Joblib, making it ready for deployment in the next phase.

Additionally, we trained our PCA model to create the T-matrix, which was also stored for later use.

We also defined the main functions that would be used throughout the project.

Below, we present the Python implementation :

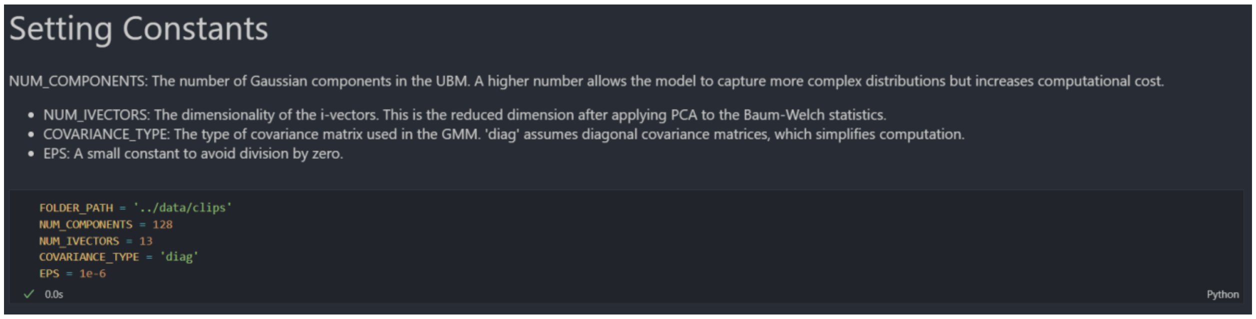


Importing Libraries

These libraries are used for data manipulation (pandas, numpy), audio processing (librosa), machine learning (sklearn), and saving/loading models (joblib).

```
import pandas as pd
import numpy as np
import librosa
import os
from sklearn.preprocessing import StandardScaler
from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
import joblib
```

✓ 9.5s Python

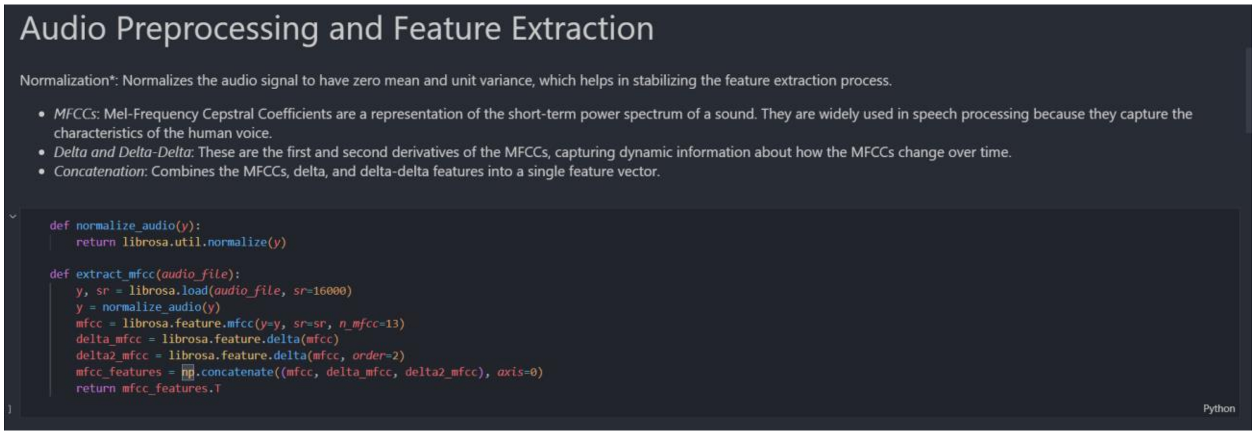

Setting Constants

NUM_COMPONENTS: The number of Gaussian components in the UBM. A higher number allows the model to capture more complex distributions but increases computational cost.

- NUM_IVECTORS: The dimensionality of the i-vectors. This is the reduced dimension after applying PCA to the Baum-Welch statistics.
- COVARIANCE_TYPE: The type of covariance matrix used in the GMM. 'diag' assumes diagonal covariance matrices, which simplifies computation.
- EPS: A small constant to avoid division by zero.

```python
FOLDER_PATH = '../data/clips'
NUM_COMPONENTS = 128
NUM_IVECTORS = 13
COVARIANCE_TYPE = 'diag'
EPS = 1e-6
```

✓ 0.0s Python

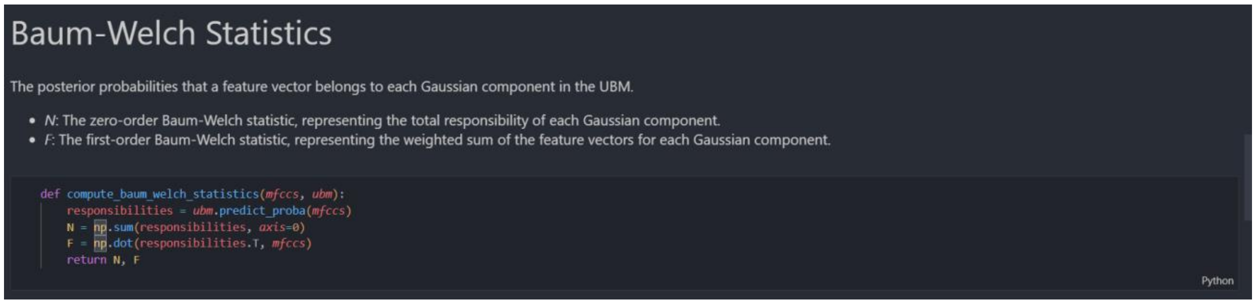

Audio Preprocessing and Feature Extraction

Normalization*: Normalizes the audio signal to have zero mean and unit variance, which helps in stabilizing the feature extraction process.

- *MFCCs*: Mel-Frequency Cepstral Coefficients are a representation of the short-term power spectrum of a sound. They are widely used in speech processing because they capture the characteristics of the human voice.
- *Delta and Delta-Delta*: These are the first and second derivatives of the MFCCs, capturing dynamic information about how the MFCCs change over time.
- *Concatenation*: Combines the MFCCs, delta, and delta-delta features into a single feature vector.

```python
def normalize_audio(y):
    return librosa.util.normalize(y)

def extract_mfcc(audio_file):
    y, sr = librosa.load(audio_file, sr=16000)
    y = normalize_audio(y)
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
    delta_mfcc = librosa.feature.delta(mfcc)
    delta2_mfcc = librosa.feature.delta(mfcc, order=2)
    mfcc_features = np.concatenate((mfcc, delta_mfcc, delta2_mfcc), axis=0)
    return mfcc_features.T
```

Python

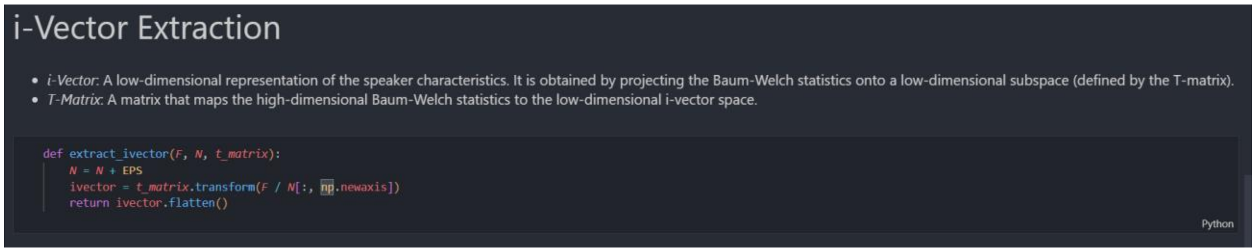

Baum-Welch Statistics

The posterior probabilities that a feature vector belongs to each Gaussian component in the UBM.

- *N*: The zero-order Baum-Welch statistic, representing the total responsibility of each Gaussian component.
- *F*: The first-order Baum-Welch statistic, representing the weighted sum of the feature vectors for each Gaussian component.

```python
def compute_baum_welch_statistics(mfccs, ubm):
    responsibilities = ubm.predict_proba(mfccs)
    N = np.sum(responsibilities, axis=0)
    F = np.dot(responsibilities.T, mfccs)
    return N, F
```

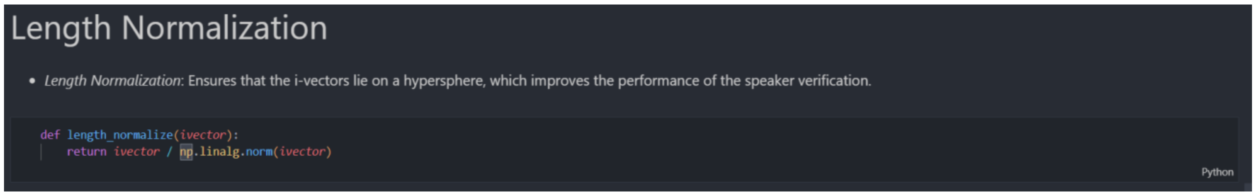
Python


i-Vector Extraction

- *i-Vector*: A low-dimensional representation of the speaker characteristics. It is obtained by projecting the Baum-Welch statistics onto a low-dimensional subspace (defined by the T-matrix).
- *T-Matrix*: A matrix that maps the high-dimensional Baum-Welch statistics to the low-dimensional i-vector space.

```python
def extract_ivector(F, N, t_matrix):
    N = N + EPS
    ivector = t_matrix.transform(F / N[:, np.newaxis])
    return ivector.flatten()
```

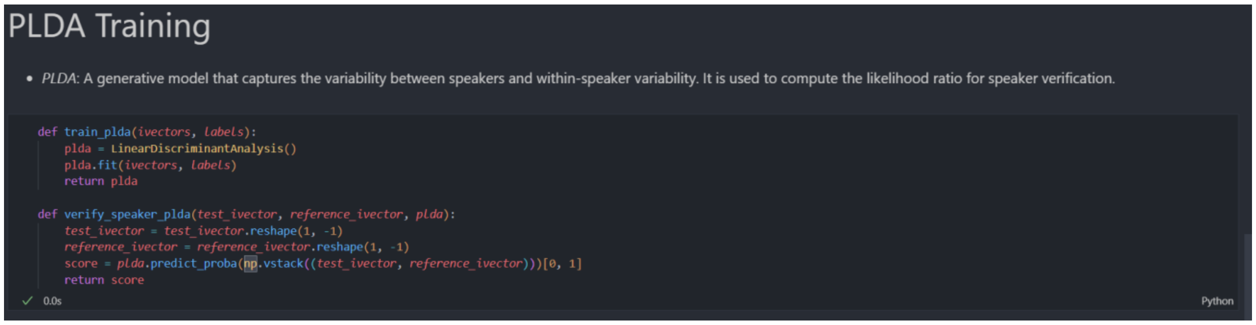
Python


Length Normalization

- *Length Normalization*: Ensures that the i-vectors lie on a hypersphere, which improves the performance of the speaker verification.

```python
def length_normalize(ivector):
    return ivector / np.linalg.norm(ivector)
```

Python

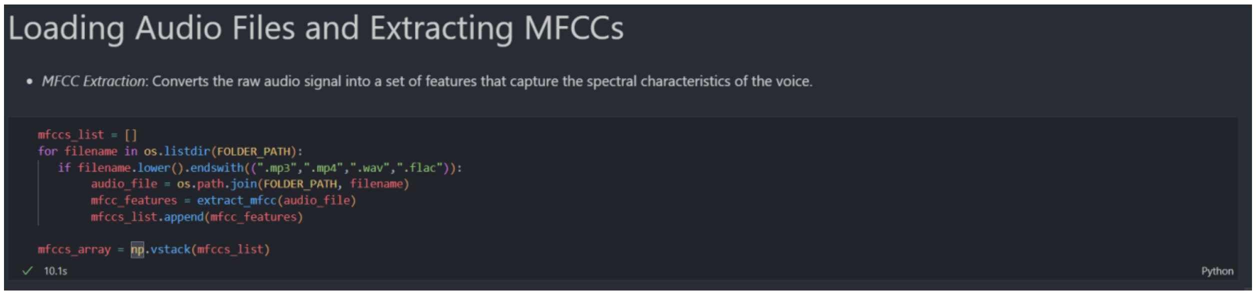


PLDA Training

- *PLDA*: A generative model that captures the variability between speakers and within-speaker variability. It is used to compute the likelihood ratio for speaker verification.

```python
def train_plda(ivectors, labels):
    plda = LinearDiscriminantAnalysis()
    plda.fit(ivectors, labels)
    return plda

def verify_speaker_plda(test_ivector, reference_ivector, plda):
    test_ivector = test_ivector.reshape(1, -1)
    reference_ivector = reference_ivector.reshape(1, -1)
    score = plda.predict_proba(np.vstack((test_ivector, reference_ivector)))[0, 1]
    return score
```

✓ 0.0s Python

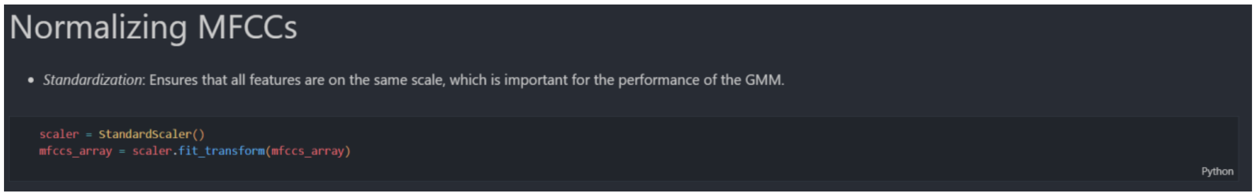


Loading Audio Files and Extracting MFCCs

- *MFCC Extraction*: Converts the raw audio signal into a set of features that capture the spectral characteristics of the voice.

```python
mfccs_list = []
for filename in os.listdir(FOLDER_PATH):
   if filename.lower().endswith((".mp3",".mp4",".wav",".flac")):
       audio_file = os.path.join(FOLDER_PATH, filename)
       mfcc_features = extract_mfcc(audio_file)
       mfccs_list.append(mfcc_features)

mfccs_array = np.vstack(mfccs_list)
```

✓ 10.1s Python

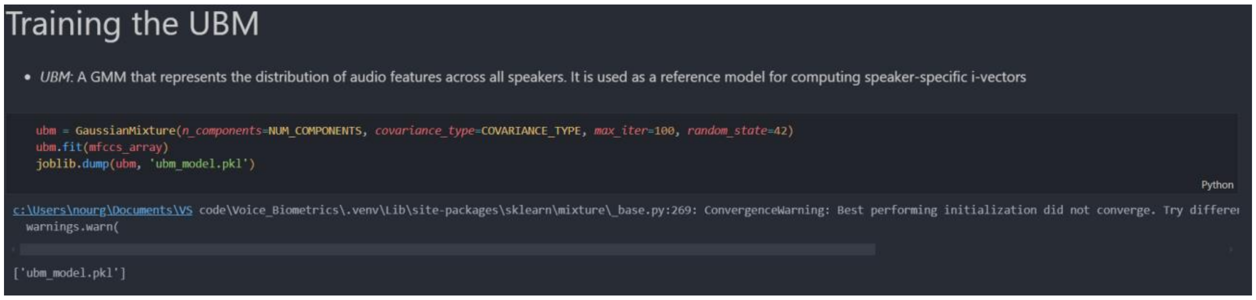


Normalizing MFCCs

- *Standardization*: Ensures that all features are on the same scale, which is important for the performance of the GMM.

```python
scaler = StandardScaler()
mfccs_array = scaler.fit_transform(mfccs_array)
```

Python



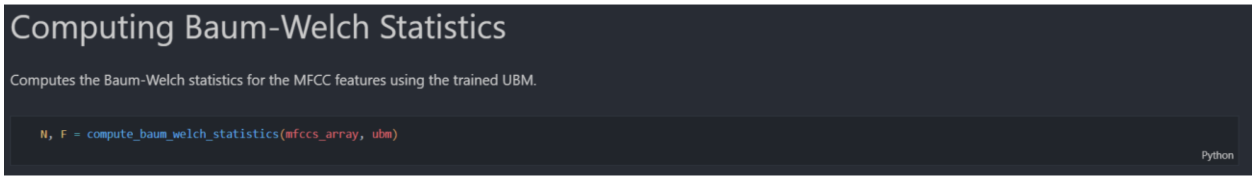
Training the UBM

- *UBM*: A GMM that represents the distribution of audio features across all speakers. It is used as a reference model for computing speaker-specific i-vectors

```python
ubm = GaussianMixture(n_components=NUM_COMPONENTS, covariance_type=COVARIANCE_TYPE, max_iter=100, random_state=42)
ubm.fit(mfccs_array)
joblib.dump(ubm, 'ubm_model.pkl')
```

Python

```
c:\Users\nourg\Documents\VS code\Voice_Biometrics\.venv\Lib\site-packages\sklearn\mixture\_base.py:269: ConvergenceWarning: Best performing initialization did not converge. Try differe
  warnings.warn(

['ubm_model.pkl']
```

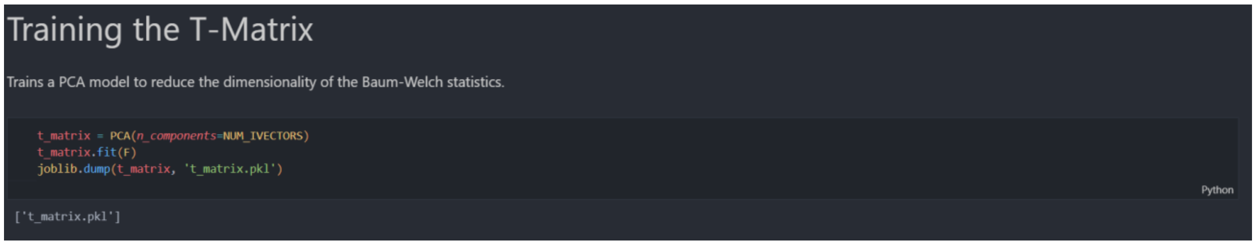


Computing Baum-Welch Statistics

Computes the Baum-Welch statistics for the MFCC features using the trained UBM.

```python
N, F = compute_baum_welch_statistics(mfccs_array, ubm)
```

Python



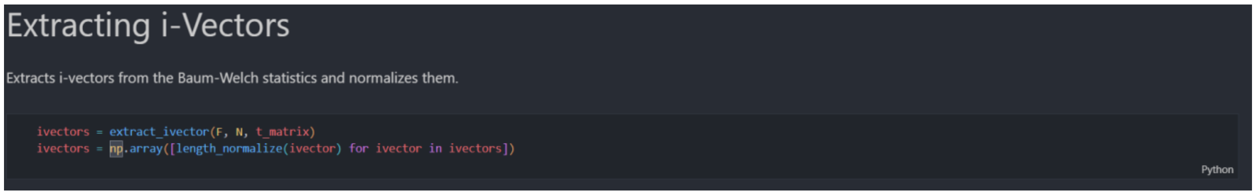
Training the T-Matrix

Trains a PCA model to reduce the dimensionality of the Baum-Welch statistics.

```python
t_matrix = PCA(n_components=NUM_IVECTORS)
t_matrix.fit(F)
joblib.dump(t_matrix, 't_matrix.pkl')
```

Python

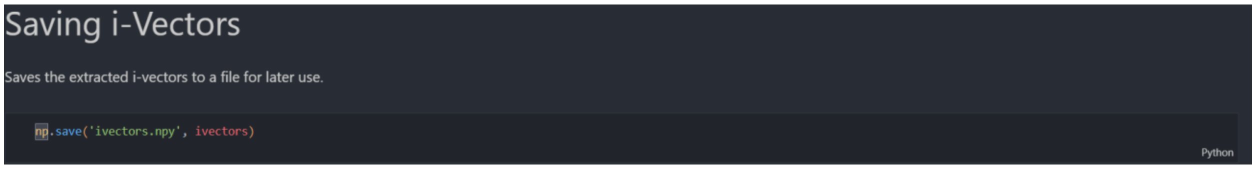
```
['t_matrix.pkl']
```


Extracting i-Vectors

Extracts i-vectors from the Baum-Welch statistics and normalizes them.

```python
ivectors = extract_ivector(F, N, t_matrix)
ivectors = np.array([length_normalize(ivector) for ivector in ivectors])
```

Python


Saving i-Vectors

Saves the extracted i-vectors to a file for later use.

```python
np.save('ivectors.npy', ivectors)
```

Python

Testing Phase :

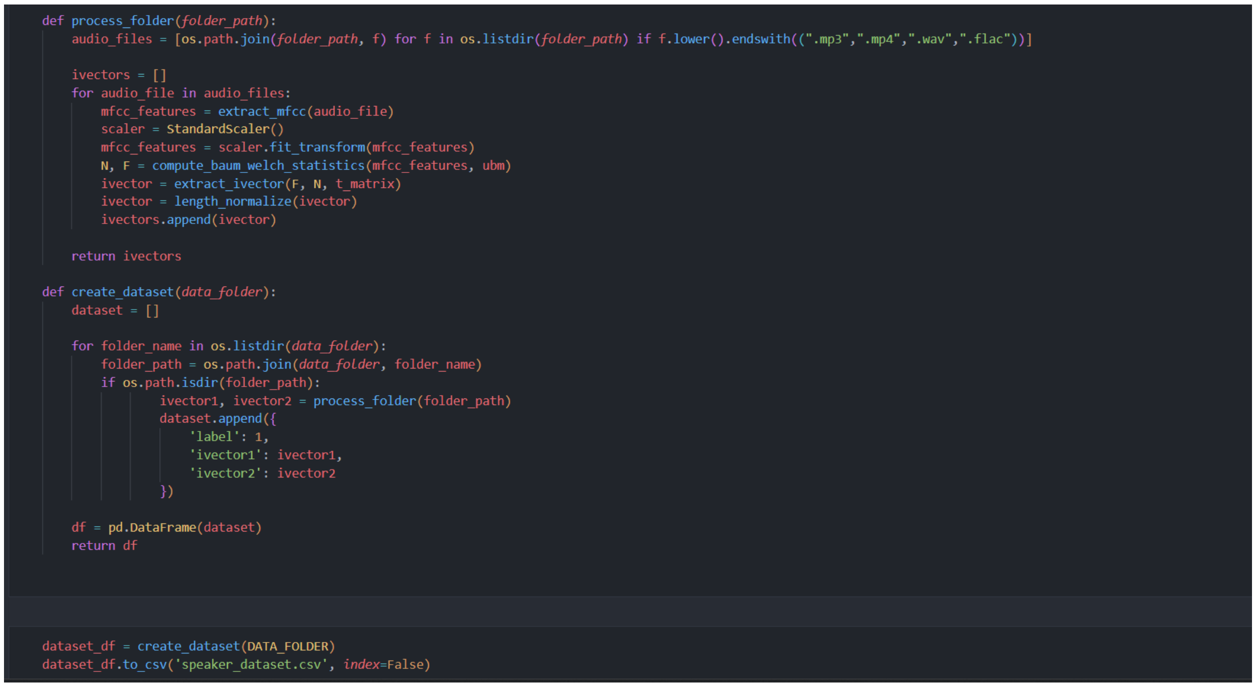
In this phase, we use our pre-trained UBM and T-matrix to generate i-vectors from test data. For this, we used the VoxCeleb dataset, a well-known dataset for voice verification, as it allows us to compare two voice clips and determine whether they belong to the same person or not. This is essentially a machine learning classification task based on i-vectors.

To classify the i-vectors, we can use any machine learning classifier. In this project, we used a Random Forest classifier, but we could also use PLDA (Probabilistic Linear Discriminant Analysis). The score obtained from this classification represents the performance of our models, including both the UBM and the classifier itself.

At this stage, we can apply data augmentation or fine-tune our models to improve performance. Additionally, we can analyze the classifier's results in detail using evaluation metrics such as the confusion matrix, recall, precision, and accuracy, along with other classification-related techniques.

Below, we present the Python implementation :

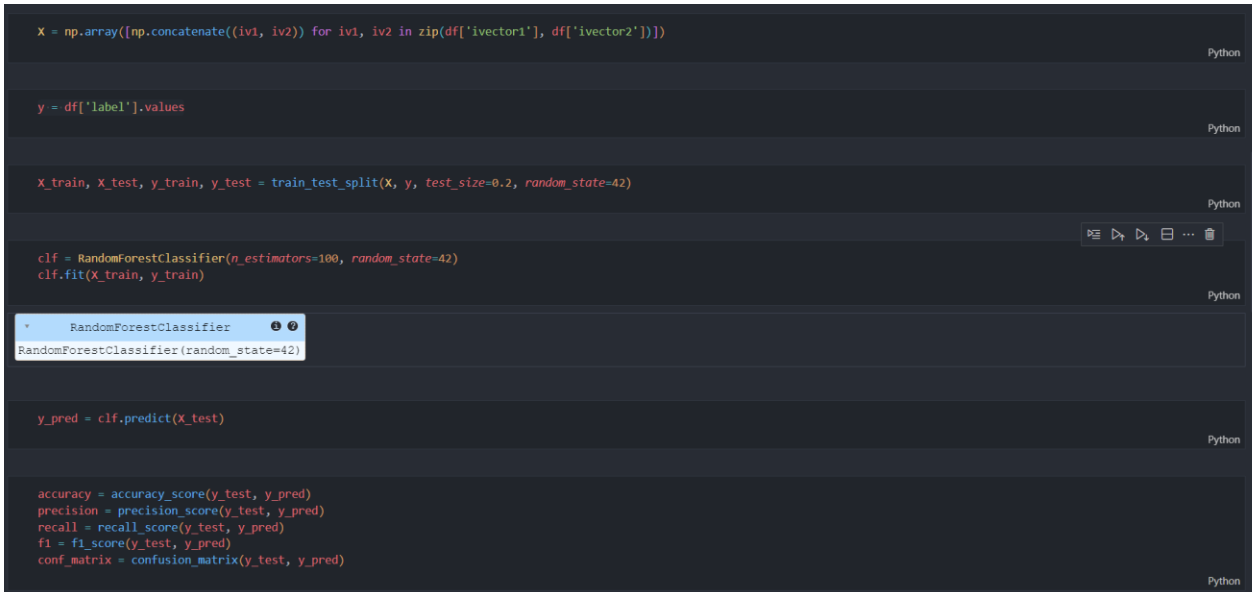
1. **We calculate the i-vectors for the entire VoxCeleb dataset and store them in a CSV file.**



```python
def process_folder(folder_path):
    audio_files = [os.path.join(folder_path, f) for f in os.listdir(folder_path) if f.lower().endswith((".mp3",".mp4",".wav",".flac"))]

    ivectors = []
    for audio_file in audio_files:
        mfcc_features = extract_mfcc(audio_file)
        scaler = StandardScaler()
        mfcc_features = scaler.fit_transform(mfcc_features)
        N, F = compute_baum_welch_statistics(mfcc_features, ubm)
        ivector = extract_ivector(F, N, t_matrix)
        ivector = length_normalize(ivector)
        ivectors.append(ivector)

    return ivectors

def create_dataset(data_folder):
    dataset = []

    for folder_name in os.listdir(data_folder):
        folder_path = os.path.join(data_folder, folder_name)
        if os.path.isdir(folder_path):
                ivector1, ivector2 = process_folder(folder_path)
                dataset.append({
                    'label': 1,
                    'ivector1': ivector1,
                    'ivector2': ivector2
                })

    df = pd.DataFrame(dataset)
    return df
```

```python
dataset_df = create_dataset(DATA_FOLDER)
dataset_df.to_csv('speaker_dataset.csv', index=False)
```

2. **We train and test our Random Forest classifier on the data we processed earlier.**



```python
X = np.array([np.concatenate((iv1, iv2)) for iv1, iv2 in zip(df['ivector1'], df['ivector2'])])
```
Python

```python
y = df['label'].values
```
Python

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```
Python

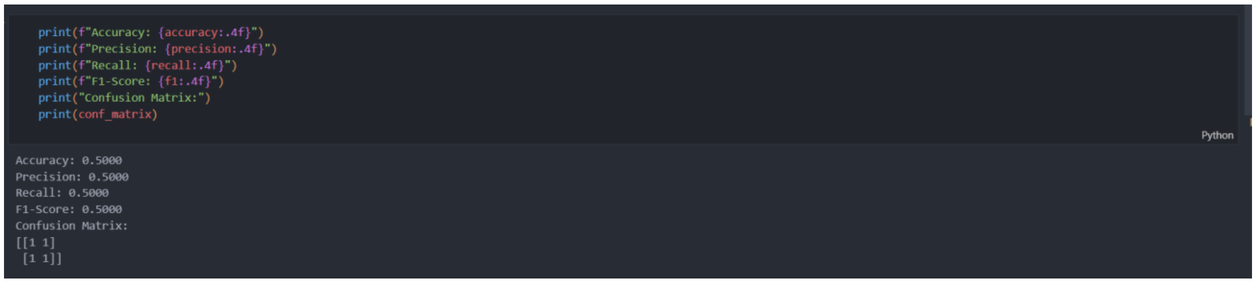
```python
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
```
Python

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```python
y_pred = clf.predict(X_test)
```
Python

```python
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
```
Python

3. **Results and scores.**



```python
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
```

```
Accuracy: 0.5000
Precision: 0.5000
Recall: 0.5000
F1-Score: 0.5000
Confusion Matrix:
[[1 1]
 [1 1]]
```

Conclusions And Discussions

Discussions :

After testing, the results were not as good as expected. We achieved an F1 score of 0.5, which indicates that our model performs no better than random guessing essentially like flipping a coin with a 50/50 probability. This level of performance is far from ideal, and several factors contributed to this outcome.

The primary issue was the lack of data. Due to time constraints and computational limitations, I was unable to obtain a large dataset like VoxCeleb, which requires downloading and processing over 15GB of data. On my current hardware, this process would have taken days or even weeks. As a result, the model was trained on a small dataset, which significantly limited its ability to generalize and make accurate predictions.

Another challenge is the computational cost of statistical methods like Gaussian Mixture Models (GMMs) and i-vectors. These methods are both time and memory intensive, making them less practical for large-scale applications without significant computational resources.

To address the issue of limited data, techniques like data augmentation could be employed. For example, we could modify acoustic parameters such as pitch and fundamental frequency (F0) to generate additional voice samples from the existing dataset. However, even with these improvements, statistical methods remain computationally expensive compared to modern deep learning approaches.

Future Improvements :

In the future, I plan to invest more time in addressing the lack of data issue. One approach is to explore deep learning techniques like x-vectors, which can uncover complex patterns in human voices and significantly improve the accuracy and security of speaker verification systems.

While statistical methods like i-vectors can achieve high performance, they come with a trade-off in terms of time and computational resources. Deep learning models, on the other hand, offer a more scalable and efficient solution, especially when trained on large datasets.

Future Vision :

My long-term vision is to improve the performance of the system and create a preprocessing pipeline that encapsulates all the code and methods used. This pipeline will be crucial for deploying the system in real-world applications.

Next, I aim to develop a web interface where users can sign up and log in using their voice as a password (voice authentication). To enhance security, I will adopt a text-independent approach, where users are prompted to read a randomly generated phrase each time they log in. This approach will help prevent spoofing attacks, which were common in older systems that relied on text-dependent methods (where users always read the same phrase).

Additionally, I will integrate an Automatic Speech Recognition (ASR) system to verify that users are reading the prompted phrase correctly. This will add an extra layer of security and ensure the authenticity of the voice samples.

Conclusion :

In conclusion, this project provided valuable insights into the challenges and complexities of building a speaker verification system. While the results were not as strong as hoped, they highlighted the importance of data quality and quantity in training robust models. The limitations of statistical methods, such as their computational cost and sensitivity to small datasets, underscore the need for more advanced techniques like deep learning.

Moving forward, I plan to explore deep learning-based approaches like x-vectors and ECAPA-TDNN, which offer greater accuracy and scalability. Additionally, I aim to develop a user-friendly web interface for voice authentication, incorporating text-independent prompts and ASR to enhance security.

Despite the challenges, this project has been a rewarding learning experience, and I am excited to continue improving the system and exploring new possibilities in the field of voice biometrics.

References :

1. Yu, Dong, and Li Deng. Automatic Speech Recognition: A Deep Learning Approach. Springer, 2015.

2. Kamath, Uday, John Liu, and James Whitaker. Deep Learning for NLP and Speech Recognition. Springer, 2019.

3. Beigi, Homayoon. Fundamentals of Speaker Recognition. Springer, 2011.

4. García-Mateo, Carmen, and Gérard Chollet, editors. Voice Biometrics: Technology, Trust, and Security. Springer, 2014.