Voice-Based Re-Identification

Biometric Systems Project - University of Rome La Sapienza, AY 2023-2024

Davide Belcastro Matricola: 1962536
Susanna Cifani Matricola: 1953238

December 26, 2023

1 Introduction

A biometric system uses an individual's unique biological or behavioral characteristics for identification and authentication. Among the various biometric modalities, voice reidentification is based on analyzing the unique characteristics of a person's voice to identify them in different contexts.
"The human voice is an exceptional medium of communication. It cannot be lost, forgotten, or acquired without the conscious participation of its holder. Due to complicated processes, voice production strictly depends on the anatomy of the vocal organs, and the speech organ features significant ontogenetic differences. The voice as a biometric ID can be recorded by means of commonly used electronic devices, and its measurement is socially acceptable as it does not require direct contact. Nowadays, 'touchless' biometric IDs and solutions are extremely important and are still gaining importance during the time of crisis situations such as the COVID-19 pandemic. The current pandemic situation makes it impossible to verify identity by means of e.g., fingerprints, due to the common use of protective gloves. Face recognition is also hindered as people wear masks that cover a major part of the face. Thus, voice biometrics seems to be a good solution to that problem, as the protective elements mentioned above are irrelevant with respect to this method of biometric identification."
References here.
We can see the fig. 1

Figure 1: Advantage of voice biometrics over face and fingerprint biometrics during COVID-19 pandemic.

1.1 Outline

- Acquisition of the Voice Signal (download of the dataset).
- Feature Extraction.
- TrainSet and TestSet creation: The extracted information is used to create a dataset of labeled features.
- Machine Learning model: A ML model is trained on the TrainSet and all evaluation metrics on the TestSet are calculated.
- Reidentification Run-time: A user can be registered and subsequently re-identified.

2 Dataset

We downloaded the AudioMNIST dataset and used the "data" directory content.
The dataset used contains 30,000 audio samples of numbers (0...9) pronounced by 60 different speakers. Each speaker has a separate directory with their 500 audio recordings.

3 Audio Signal Converter

To extract the features from the audio samples, we used the Python library **Librosa**, which is employed for the analysis of audio files.
In fact, Librosa can load an audio file in *.wav* format and transform it into a numerical signal suitable for analysis.
The numerical signal becomes the input for the feature functions to be extracted.

```python
import librosa

# Specifica il percorso del file .wav
audio_file_path = 'percorso/del/file/audio.wav'

# Carica il file audio e ottieni il segnale numerico e il tasso di campionamento
audio_signal, sample_rate = librosa.load(audio_file_path, sr=None)
```

Listing 1: Extraction of the signal from a .wav file with Librosa

In this example, *audio_signal* represents the numerical signal from the .wav file, and *sample_rate* is the sampling rate of the signal.

4 Dataset creation

For each numerical signal, the most relevant features are extracted. Subsequently, these features are used to create a CSV file. In this file, each row represents a numerical signal, with the extracted features as columns and the associated user label.

4.1 Feature extraction

The extracted features are:

- **MFCCs (Mel-frequency cepstral coefficients)**
  Description: Represent the short-term spectral content of a sound, dividing it into frequency bands. Cepstral coefficients capture the most relevant features of the audio signal.
  Importance: Widely used in voice pattern identification, effectively representing human sound characteristics.
  Output: 13-element vector.
  Example Python:

```python
    mfccs = librosa.feature.mfcc(y=audio_signal, sr=sample_rate, n_mfcc=13)

```

- **Spectral centroid**
  Description: Represents the "center of mass" of the sound spectrum, indicating where most of the spectral energy is located.
  Importance: Provides information about the predominant pitch of the sound, helping to distinguish between sharp and flat sounds.
  Output: Each audio returns a vector of different sizes; padding is performed, and the size of the maximum size vector is calculated. Zeros are added to all other returned vectors to bring them to the same dimension.
  Example Python:

```python
    spectral_centroid = librosa.feature.spectral_centroid(y=audio_signal, sr=sample_rate)

```

- **Chroma**
  Description: Captures the energy distribution in pitch classes, helping to identify the harmonic content of an audio signal.
  Importance: Useful for recognizing musical chords and distinguishing between different types of harmonic sounds.
  Output: Returns a 12-element vector.
  Example Python:

```python
    chroma = librosa.feature.chroma_stft(y=audio_signal, sr=sample_rate)

```

- **Zero Crossings**
  Description: Counts the rate of sign changes in the audio signal, providing information about sound noisiness.
  Importance: Can be indicative of sound complexity and contribute to distinguishing between clearer and noisier sounds.
  Output: Returns a value.
  Example Python:

```python
    zero_crossings = librosa.feature.zero_crossing_rate(audio_signal)

```

- **RMS energy**
  Description: Measures the "volume" of the sound, i.e., the overall energy present in the signal.
  Importance: Useful for characterizing the intensity of the sound and can be used to distinguish between loud and soft sounds.
  Output: Returns a value.
  Example Python:

```python
    rms_energy = librosa.feature.rms(y=audio_signal)

```

- **Spectral contrast**
  Description: Measures the amplitude difference between peaks and valleys in the spectrum, capturing perceptual features of the sound.
  Importance: Helps identify significant differences in the spectral structure of the sound.
  Output: Returns a 7-element vector.
  Example Python:

```python
    spectral_contrast = librosa.feature.spectral_contrast(y=audio_signal, sr=sample_rate)

```

- **Spectral bandwidth**
  Description: Measures the width of the spectrum, providing information about the spread of frequencies in the signal.
  Importance: Can be useful for distinguishing between sounds with different frequency distributions.
  Output: Each audio returns a vector of different sizes; padding is performed, the maximum size vector is calculated, and zeros are added to all other vectors returned by this function to bring them to the same dimension.
  Example Python:

```
spectral_bandwidth = librosa.feature.spectral_bandwidth(y=audio_signal, sr=sample_rate)
```

- **Spectral rolloff**
  Description: Indicates the frequency below which a specified percentage of the total spectral energy is located.
  Importance: Can be indicative of the overall "brightness" of the sound, helping to distinguish between brighter and duller sounds.
  Output: Returns a value.
  Example Python:

```
spectral_rolloff = librosa.feature.spectral_rolloff(y=audio_signal, sr=sample_rate)
```

- **Spectral flatness**
  Description: Measures how flat or peaked the spectrum is, indicating the noisiness of the sound.
  Importance: Can be used to recognize more uniform sounds or sounds richer in harmonics.
  Output: Returns a value.
  Example Python:

```
spectral_flatness = librosa.feature.spectral_flatness(y=audio_signal)
```

These features provide crucial information about the structure and spectral content of an audio signal, making them valuable for the identification and recognition of vocal patterns in machine learning models dedicated to speech recognition and sound analysis. This allows the system to distinguish between different speakers.
Padding is performed to normalize the data. In machine learning algorithms, this phase is essential for the system to work correctly with the data.

5 Proposed methods

- `With PCA:` A dimensionality reduction was performed using Principal Component Analysis (PCA). Specifically, the analysis of principal components was applied to reduce the dimensionality of the dataset while retaining the most significant information.

  Firstly, the StandardScaler class was utilized to standardize the data in the training set X_train. The standardization process scales the data to have a mean of zero and a standard deviation of one. This step was necessary to ensure that all features had the same scale, preventing some features from having a dominant weight during the analysis.

  PCA was calculated without specifying the number of desired components, allowing us to obtain all principal components. To understand the amount of variance explained by each principal component, a cumulative variance graph was created, as shown in Fig. 2. Using the explained_variance_ratio_ method of the PCA model, the ratio of the variance explained by each principal component to the total variance was calculated. Subsequently, the cumulative trend of this ratio was computed and represented in the graph. The x-axis of the graph represents the number of principal components, while the y-axis represents the cumulative explained variance.

  In the graph, a vertical blue line was drawn to represent the number of principal components selected as the optimal value for dimensionality reduction, in this case, 26 components. It is essential to choose a number of principal components that explains at least 95% of the total dataset variance, as indicated by the red horizontal line in the graph.

  Once the optimal number of principal components was determined, dimensionality reduction was performed by applying PCA with the parameter n_components=26.

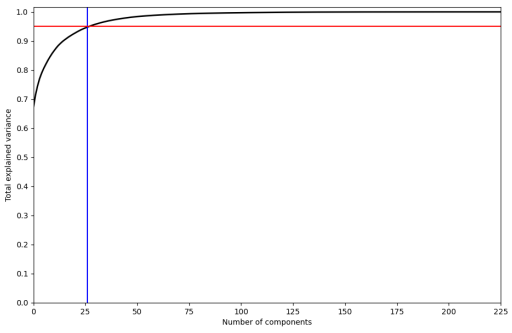


Figure 2: Principal component analysis

In conclusion, the analysis of cumulative variance using PCA identified the optimal number of principal components needed to explain a significant amount of variance in the dataset, allowing a reduction in the computational time required for training (from 4-5 hours to less than an hour). However, this comes with a slight decrease in the performance of the trained model.

- `Without PCA:` All the extracted features were used to train the model, which took several hours, however obtaining slightly more satisfactory results compared to using PCA with 26 principal components.

We developed the method both using all the extracted features (without PCA) and with a reduction of the dimensionality of the problem by applying PCA.

Finally we compared all the results obtained from the different methods.

5.1 Dataset Division

The split involves creating two directories, *training* and *testing*, with 80% of each user's audio files in training and 20% in testing. Two approaches have been implemented:

- Single Training
  The training and testing split is performed only once.

- K Cross-Validation
  The training and testing split is performed k times.

5.2 Model and Training

A Random Forest model has been trained on the training files. In k cross-validation, k Random Forest models are generated and combined with an **Stacking Ensemble**.

5.2.1 Stacking Ensemble

Ensemble stacking is a technique in machine learning where different models are combined to improve the overall performance of the system. The goal is to leverage the diversity between models to achieve better results than a single model.

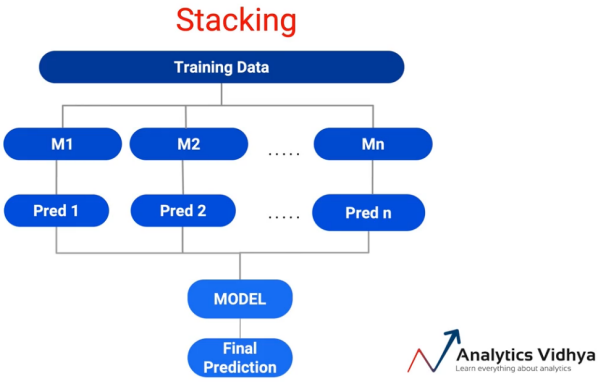


Figure 3: Stacking Ensemble

5.3 Testing Types

The accuracy is calculated in three different ways. A file named 'accessi.csv' is created, in which the columns are yTest (the actual user) and the rf_predictions, where rf_predictions[i] is the probability that the user is i.

- **Identification Closed Set**
  All elements of yTest belong to the dataset.

- **Identification Open Set**
  yTest also has indices that do not belong to the dataset, "-2" in this case.

- **Verification**
  There's also a column in which we have the declared user (who claims to be). User declarations are random: 50% of the time, the user declares the truth, and 50% of the time, the user declares a falsehood, claiming to be a user belonging to the dataset but different from the actual user.

5.3.1 Used Metrics

Brief description of the metrics used, all of which depend on the adopted acceptance (*threshold*) t:

- FAR (*False Acceptance Rate*):
  Probability of incorrectly recognizing an unregistered person as a registered person.

$$FAR = \frac{False\ Acceptance}{Total\ Acceptance}$$

- FRR (*False Rejection Rate*):
  Probability of incorrectly recognizing a registered person as an unregistered person.

$$FRR = \frac{\text{False Rejection}}{\text{Total Rejection}}$$

- ERR (*Equal Error Rate*):
  The point where FAR and FRR intersect. A device with a lower EER is considered more accurate.

- Confusion Matrix:
  The confusion matrix is a tool used to evaluate the performance of a classification model. It consists of a table where the columns are *Predicted Negative* and *Predicted Positive*, while the rows are *Real Negative* and *Real Positive*, comparing the model's predictions with the actual classes of the dataset. Below is a generic representation of a confusion matrix:

	PN	PP
RN	TN	FP
RP	FN	TP

  - **FP (False Positive):**
    The model incorrectly predicted that an instance belongs to the positive class when it actually belongs to the negative class (false alarm).
  - **FN (False Negative):**
    The model incorrectly predicted that an instance belongs to the negative class when it actually belongs to the positive class (missed detection).
  - **TN (True Negative):**
    The model correctly predicted that an instance belongs to the negative class.
  - **TP (True Positive):**
    The model correctly predicted that an instance belongs to the positive class.

  The confusion matrix provides a detailed analysis of the model's performance, allowing the calculation of various evaluation metrics such as precision, recall, and F1 score.

- Precision:
  Precision measures the accuracy of positive predictions made by the model and is calculated as:

$$Precision = \frac{\text{TP}}{\text{TP+FP}}$$

- Recall (Sensitivity):
  Recall measures the ability of the model to capture all positive instances and is calculated as:

$$Recall = \frac{TP}{TP + FN}$$

- F1 Score:
  F1 Score is the harmonic mean of precision and recall, providing a balance between the two metrics, and is calculated as:

$$F1Score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

- DET (*Detection Error Trade-off*):
  Graph of error rates for binary classification systems that correlates False Acceptance Rate with False Rejection Rate.

- ROC (*Receiver Operating Characteristic*):
  Curve plotted on a Cartesian plane where the axes are Genuine Acceptance Rate and False Acceptance Rate. It represents the probability of correctly identifying a subject as the FAR varies.

6 Results

We illustrate the results obtained with different approaches.

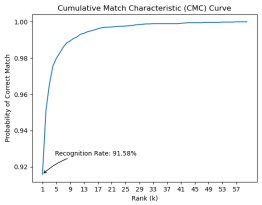
6.1 Without PCA

6.1.1 Identification Close-set

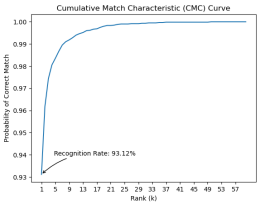
The **Cumulative Match Characteristic (CMC)** is a curve plotted on a Cartesian plane with the Probability of Identification (y-axis) and Rank k (x-axis). It represents the probability of correctly identifying a subject as the Rank k varies.

Of particular interest is the *Recognition Rate*, which denotes the probability of correct identification at Rank 1.

In the graphs 4a and 4b, we can observe that applying 5 Cross Validations leads to a higher Recognition Rate.



(a) CMC ST Without PCA, close-set



(b) CMC CV Without PCA, close-set

Precision, Recall, and F1 Score are also slightly higher, and the results are presented in the table 1.

Value	Single Training	5 Cross Validation
Precision	0.916	0.9321
Recall	0.9158	0.9311
F1 Score	0.916	0.9316

Table 1: Tab. Accuracy without PCA, close-set

6.1.2 Identification Open-set

The **Open-set (Watchlist) Receiver Operating Characteristic (ROC)** depicts DIR against FAR and is useful for finding the optimal threshold, balancing the maximization of DIR and the minimization of FAR. A compromise is necessary as both measures depend on the threshold, but in opposite directions:

- If we raise the threshold, FAR decreases, but DIR also decreases.
- If we lower the threshold, DIR increases, but FAR also increases.

We can see the ROC Curve in fig. 5 and 6.

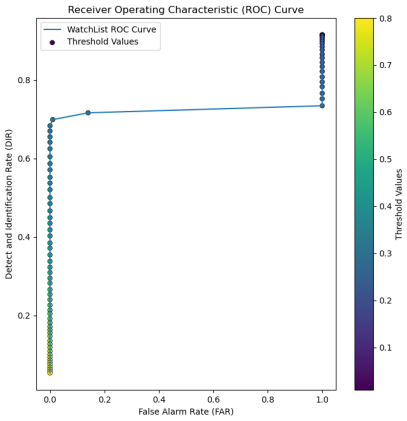


Figure 5: WR Single Training Without PCA, open-set

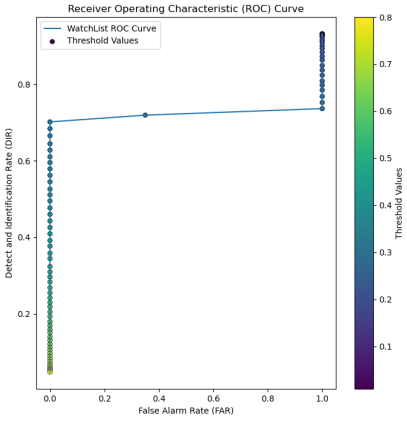
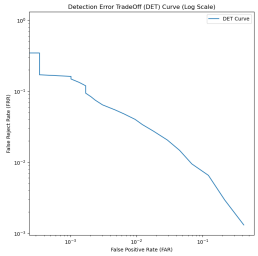


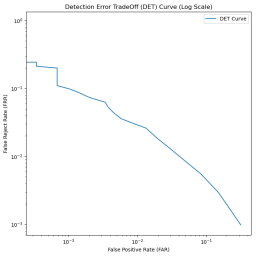
Figure 6: WR CV Without PCA, open-set

6.1.3 Verification

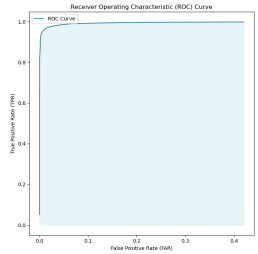
During verification, a subject is accepted if the similarity score obtained from matching with gallery models corresponding to the declared identity is $\geq$ the threshold value; otherwise, it is rejected. We illustrate in Figs. 7a, 7e, 7c, and 7g the graphs generated by Single Training verification. Meanwhile, in Figs. 7b, 7f, 7d, and 7h, the graphs are generated by 5 Cross Validation. The Confusion Matrices in Figs. 7g and 7h were generated by extracting the ideal threshold from their respective FAR and FRR curves 7e and 7f.
It can be observed that the ERR rate is slightly lower when applying 5 Cross Validation. Also, the Confusion Matrix of 5 Cross Validation has higher values in TN and TP and lower values in FN and FP compared to Single Training.



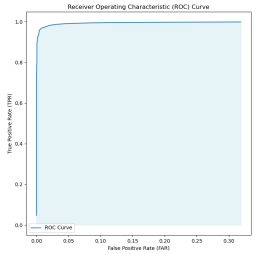
(a) DET ST, Verification



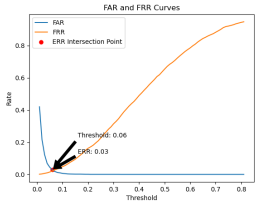
(b) DET CV, Verification



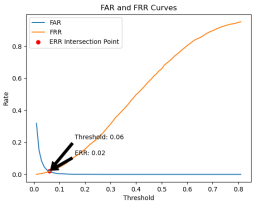
(c) ROC ST, Verification



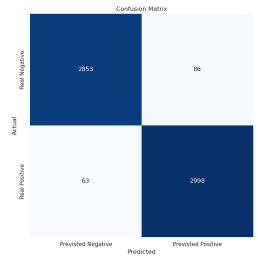
(d) ROC CV, Verification



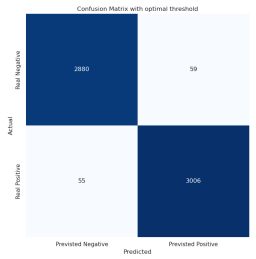
(e) FAR and FRR ST, Verification



(f) FAR and FRR CV, Verification



(g) Confusion Matrix ST, Verification



(h) Confusion Matrix CV, Verification

6.2 PCA

The results were generated by applying 10 cross validations

- Identification Close-set
  We can see the CMC in fig.8

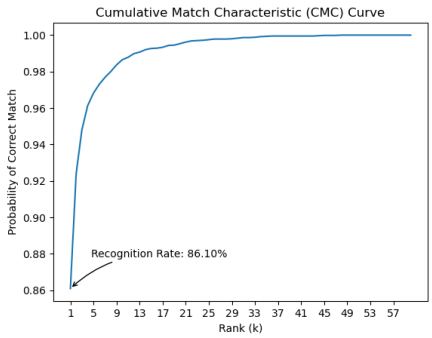


Figure 8: CMC with PCA

Other calculated parameters:

  - **Precision:** 0.8639
  - **Recall:** 0.861
  - **F1-Score:** 0.860

- Verification
  We illustrate the Confusion Matrix in fig. 9 created by extrapolating the ideal threshold from the FAR and FRR curve.

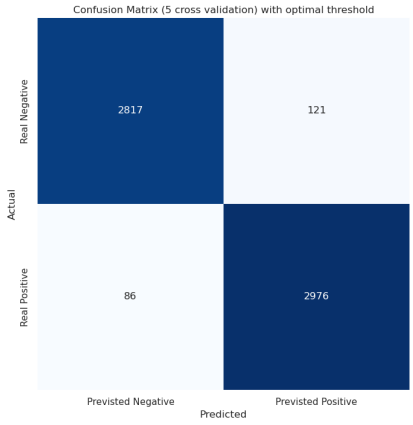


Figure 9: Confusion Matrix with PCA

7 Re-Identification personal test

A demo has been implemented where a user can register in the system and be re-identified later. We have included various voices of some users, including ourselves, in the dataset. If the application outputs a prediction value greater than 0.8, the method performs template updating: the extracted features are inserted into the dataset, and the model is retrained.

In Fig. 10, an example execution is illustrated. The user can register in the system by pressing the y button. In this case, the user needs to follow the registration procedure, which involves pronouncing certain phrases. If the user presses the n button, they will be re-identified. If the probability of re-identification is greater than 0.8, the features computed for re-identification will be inserted into the dataset, and the model will be retrained (template updating).

If the user presses the z button, additional records will be added to the dataset by pronouncing additional phrases. After this process, the model is retrained once again.

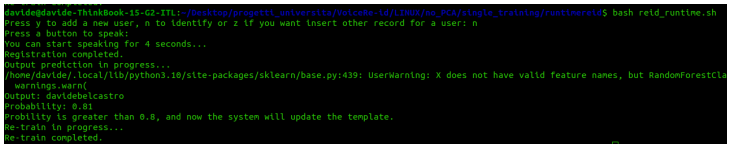


Figure 10: re-id runtime

8 Conclusions

We have analyzed the results obtained in detail, focusing on:

- Identification open-set
- Identification close-set
- Verification

and we have observed that by applying Cross Validation with 5 folds, the results are slightly better compared to single training. It would be interesting to find the limit beyond which increasing k continues to improve accuracy without causing overtraining and overfitting.

By applying PCA, the results are inferior, but the training phase is significantly faster.

The system can be further improved by identifying additional meaningful features in voice recognition.

9 References

The project is available on GitHub.