University of Engineering & Technology, Lahore

Department of Electrical Engineering

Training of System for Real Time Speaker Identification using PLP & MFCC feature extraction and SVM & GMM-UBM Classifiers

---

Arooj Fatima (2020-EE-152)

Aziz Haider (2020-EE-172)

Subhan Mansoor (2020-EE-175)

EE-439: Introduction to Machine Learning

CEP Report || May 24, 2023

Contents of Report

Abstract

The aim of this study is to compare the performance of two feature extraction methods, namely Perceptual Linear Prediction (PLP) and Mel-Frequency Cepstral Coefficients (MFCC), in combination with two classification algorithms, Support Vector Machines (SVM) and Gaussian Mixture Model-Universal Background Model (GMM-UBM), for real-time speaker identification. The study uses a dataset consisting of 9 recordings of students as training & test sets to train and test the system. Results show that the combination of MFCC and SVM achieves the highest accuracy of 83.5% on the test set.

Introduction

Real-time speaker identification has become an important field of research due to its potential applications in various domains such as healthcare, education, and entertainment. Speech recognition systems aim to transcribe spoken language into text, which can then be used for further analysis or processing. One of the key components of speech recognition systems is feature extraction, which involves transforming raw audio signals into a set of representative features that can be used for classification.

In this study, we compare the performance of two widely used feature extraction methods, PLP and MFCC, in combination with two classification algorithms, SVM and GMM-UBM, for real-time speech recognition. The study uses a dataset consisting of 9 recordings of students for the training of the system. The study aims to determine which combination of feature extraction method and classification algorithm yields the highest accuracy in speech recognition.

Feature Extraction Methods

1. Mel-Frequency Cepstral Coefficients (MFCC)

Mel Frequency Cepstral Coefficients (MFCC) are a feature extraction method that is based on the mel frequency scale. The mel frequency scale is a nonlinear transformation of the frequency scale that is more closely aligned to the way that humans perceive pitch.

- Preprocess the audio signal by applying pre-emphasis to balance the spectrum and amplify higher frequencies. Divide the pre-emphasized signal into frames with overlap to analyze small segments of the audio.
- Apply a windowing function, such as the Hamming window, to each frame to reduce spectral leakage. Compute the magnitude spectrum using the Fast Fourier Transform (FFT) for each windowed frame.
- Create a set of Mel filters that cover the desired frequency range, typically around 20-40 filters. Apply the Mel filters to the magnitude spectrum to obtain filterbank energies representing energy distribution across different frequency bands.
- Perform logarithmic compression on the filterbank energies to mimic the non-linear human perception of sound.
- Apply the Discrete Cosine Transform (DCT) to the logarithmically compressed filterbank energies. Extract a smaller set of coefficients, typically the first few cepstral coefficients, capturing relevant information about the spectral envelope.

- Normalize the resulting MFCC coefficients by subtracting the mean and dividing by the standard deviation.
- The resulting MFCC coefficients serve as representative features for further analysis or modeling in speech and audio processing tasks.

2. Perceptual Linear Prediction (PLP)

Perceptual Linear Prediction (PLP) is a feature extraction method, quite similar to MFCC that is based on the way that humans perceive sound. Here's how the whole feature extraction process is carried out

- Preprocess the audio signal by applying pre-emphasis to amplify higher frequencies and balance the spectrum.
- Divide the pre-emphasized signal into frames with overlap to analyze small segments of the audio. Apply a windowing function, such as the Hamming window, to each frame to reduce spectral leakage.
- Compute the magnitude spectrum using the Fast Fourier Transform (FFT) for each windowed frame.
- Create a set of Mel filters that cover the desired frequency range, typically around 20-40 filters. Apply the Mel filters to the magnitude spectrum to obtain filterbank energies representing energy distribution across different frequency bands.
- Perform logarithmic compression on the filterbank energies to mimic the non-linear human perception of sound.
- Estimate the PLP coefficients by applying linear regression to the compressed filterbank energies, capturing spectral envelope information
- Normalize the PLP coefficients by subtracting the mean and dividing by the standard deviation to achieve zero mean and unit variance.
- The resulting PLP coefficients serve as representative features for further analysis or modeling in speech and audio processing tasks.

Classification Methods

1. Support Vector Machines (SVM)

SVM is a machine learning algorithm that is commonly used for classification tasks. It works by finding the hyperplane that maximally separates the data points into different classes. SVM can be used with different kernel functions, such as linear, polynomial, and radial basis function (RBF) kernels. In this study, we use the linear kernel, which is commonly used for speech recognition tasks.

2. Gaussian Mixture Model-Universal Background Model (GMM-UBM)

GMM-UBM is a statistical modeling technique that is commonly used for speech recognition. It models the distribution of speech features using a mixture of Gaussian probability density functions (GMM), which can capture the variability of the speech features. The UBM is a model of the background noise or non-speech sounds that are present in the training data.

Methods

The dataset used in this study consists of 9 recordings of students, each containing a set of spoken words. The recordings are preprocessed to remove any background noise or artifacts using Xtrans software which creates a .tdf file, segmenting the audio into chunks of smaller ones with minimal noise. The dataset is then split into training and validation sets, with 75% of the data used for training and the remaining 25% used for test set.

Data Segmentation

Total minutes available for the model training and testing is 171 minutes. Upon processing the data, following details were extracted

Speaker	**Minutes Available for Model Training**
Ahmed Armaghan	17.89
Ali Hussain	16.62
Ayesha Farooq	23.02
Mahnoor Khalique	17.08
Maryam Zahra	23.98
Usman Amin	17.54
Askari Zaidi	17.79
Usman Hassan	16.32
Muhammad Huazaifa	20.82
Average	19.01

Importing Libraries

```
import numpy as np
import os
import librosa
from sklearn.model_selection import train_test_split
from sklearn.mixture import GaussianMixture
from sklearn.metrics import accuracy_score
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from spafe.features.rplp import plp
import sounddevice as sd
from scipy.io.wavfile import write
```

Pre-Requisite Data Input

This part of the code defines all the necessary paths to the .tdf files and .wav audio files while also defining the sampling rate and ‘classes’ dictionary with speakers id mapped to corresponding labels

```
sam_r = 8000;      # Set sample rate as 8kHz
classes = {"ahmed":0,"ali":1,"ayesha":2,"mahnoor":3,"maryam":4,"usman_amin":5,
"askari":6,"usman_hassan":7,"huzaifa":8}

tdf_files = [
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ahmed_Armaghan\\SP80644_M_SNG_L56.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ahmed_Armaghan\\SP80644_M_SNG_L70.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ali_Hussain\\SP80643_M_SNG_L55.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ali_Hussain\\SP80643_M_SNG_L70.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ali_Hussain\\SP80643_M_SNG_L57.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ayesha_Farooq\\SP80538_F_SNG_L65.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ayesha_Farooq\\SP80538_F_SNG_L66.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Mahnoor_Khalique\\SP80645_F_SNG_L18.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Mahnoor_Khalique\\SP80645_F_SNG_L55.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Mahnoor_Khalique\\SP80645_F_SNG_L70.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Mahnoor_Khalique\\SP80645_F_SNG_L56.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Maryam_Zahra\\SP80640_F_SNG_L51.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Maryam_Zahra\\SP80640_F_SNG_L54.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Maryam_Zahra\\SP80640_F_SNG_L70.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Usman_Amin\\SP80540_M_SNG_L51.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Usman_Amin\\SP80540_M_SNG_L55.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Askari_Zaidi\\SP80641_M_SNG_L55.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Askari_Zaidi\\SP80641_M_SNG_L70.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\Data\\Usman_Hassan\\SP80642_M_SNG_L51.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\Data\\Usman_Hassan\\SP80642_M_SNG_L54.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\Data\\Usman_Hassan\\SP80642_M_SNG_L70.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Muhammad_Huzaifa\\SP80539_M_SNG_L6.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Muhammad_Huzaifa\\SP80539_M_SNG_L67.tdf',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Muhammad_Huzaifa\\SP80539_M_SNG_L69.tdf'
]

audio_files = [
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ahmed_Armaghan\\SP80644_M_SNG_L56.wav',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ahmed_Armaghan\\SP80644_M_SNG_L70.wav',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ali_Hussain\\SP80643_M_SNG_L55.wav',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ali_Hussain\\SP80643_M_SNG_L70.wav',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ali_Hussain\\SP80643_M_SNG_L57.wav',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ayesha_Farooq\\SP80538_F_SNG_L65.wav',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Ayesha_Farooq\\SP80538_F_SNG_L66.wav',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Mahnoor_Khalique\\SP80645_F_SNG_L18.wav',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Mahnoor_Khalique\\SP80645_F_SNG_L55.wav',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Mahnoor_Khalique\\SP80645_F_SNG_L70.wav',
'C:\\Users\\AbriZon\\Desktop\\CEP\\Data\\Mahnoor_Khalique\\SP80645_F_SNG_L56.wav',
```

```
'C:\\Users\\AprizOn\\Desktop\\CEP\\Data\\Maryam_Zahra\\SP80640_F_SNG_L51.wav',
'C:\\Users\\AprizOn\\Desktop\\CEP\\Data\\Maryam_Zahra\\SP80640_F_SNG_L54.wav',
'C:\\Users\\AprizOn\\Desktop\\CEP\\Data\\Maryam_Zahra\\SP80640_F_SNG_L70.wav',
'C:\\Users\\AprizOn\\Desktop\\CEP\\Data\\Usman_Amin\\SP80540_M_SNG_L51.wav',
'C:\\Users\\AprizOn\\Desktop\\CEP\\Data\\Usman_Amin\\SP80540_M_SNG_L55.wav',
'C:\\Users\\AprizOn\\Desktop\\CEP\\Data\\Askari_Zaidi\\SP80641_M_SNG_L55.wav',
'C:\\Users\\AprizOn\\Desktop\\CEP\\Data\\Askari_Zaidi\\SP80641_M_SNG_L70.wav',
'C:\\Users\\AprizOn\\Desktop\\CEP\Data\\Usman_Hassan\\SP80642_M_SNG_L51.wav',
'C:\\Users\\AprizOn\\Desktop\\CEP\Data\\Usman_Hassan\\SP80642_M_SNG_L54.wav',
'C:\\Users\\AprizOn\\Desktop\\CEP\Data\\Usman_Hassan\\SP80642_M_SNG_L70.wav',
'C:\\Users\\AprizOn\\Desktop\\CEP\\Data\\Muhammad_Huzaifa\\SP80539_M_SNG_L6.wav',
'C:\\Users\\AprizOn\\Desktop\\CEP\\Data\\Muhammad_Huzaifa\\SP80539_M_SNG_L67.wav',
'C:\\Users\\AprizOn\\Desktop\\CEP\\Data\\Muhammad_Huzaifa\\SP80539_M_SNG_L69.wav'
]
```

Feature Extraction

```
MFCC = []          # empty list to store extracted MFCC features of data
PLP = []           # empty list to store extracted PLP features of data
labels = []        # empty list to store labels of data

for i in range(len(tdf_files)):
    with open(tdf_files[i], 'r') as f:
        lines = f.readlines()

    for line in lines[3:]:  # Skip the first three lines
        data = line.strip().split('\t')
        audio_file, channel, start, end, speaker, speakerType, speakerDialect, _, _,
segment, _ = data

        if (float(end) - float(start)) < 3:
            continue

        s = int(float(start) * sam_r)  # convert start time to samples
        e = int(float(end) * sam_r)  # convert end time to samples

        if not os.path.exists(audio_files[i]):
            print(f"Audio file not found: {audio_files[i]}")
            continue

        # Load the audio file
        audio, sr = librosa.load(audio_files[i], sr=sam_r)
        audio_segment = audio[s:e]
```

```
        # extract 5 MFCC coefficients from the selected audio segment and take
transpose and then compute the mean
        mfcc_features = np.mean(librosa.feature.mfcc(y=audio_segment, sr=sam_r,
n_mfcc=5).T, axis=0)
        MFCC.append(mfcc_features)

        # extract 13 cepsrtal features from the selected audio segment then compute
the mean
        plp_features = np.mean(plp(audio_segment, sam_r, 13), axis=0)
        PLP.append(plp_features)

        # extract the speaker name as the label against the data entry
        speaker = classes[speaker]
        labels.append(speaker)

# Save MFCC and PLP features and labels to numpy files
np.save('C:\\Users\\ApriZon\\Desktop\\CEP\\MFCC_features.npy', MFCC)
np.save('C:\\Users\\ApriZon\\Desktop\\CEP\\PLP_features.npy', PLP)
np.save('C:\\Users\\ApriZon\\Desktop\\CEP\\labels.npy', labels)
```

Dividing Data into Training and Test Sets

```
# 75% Training Data and 25% Test Data
# Splitting the data into training and test sets
X_train_mfcc, X_test_mfcc, y_train_mfcc, y_test_mfcc = train_test_split(MFCC, labels,
test_size=0.25)
X_train_plp, X_test_plp, y_train_plp, y_test_plp = train_test_split(PLP, labels,
test_size=0.25)
```

Training GMM-UBM Classifier on MFCC Features

```
# Train the GMM-UBM classifier
clf_gmm_mfcc = GaussianMixture(n_components=8, covariance_type='full',
max_iter=1000).fit(X_train_mfcc)

# Unsupervised Learning
# Predict labels for the test set
y_pred_mfcc = clf_gmm_mfcc.predict(X_test_mfcc)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test_mfcc, y_pred_mfcc)
print("Accuracy:", accuracy*100)
```

Predicted Accuracy of this model was 53.36%

Training GMM-UBM Classifier on PLP Features

```
# Train the GMM-UBM classifier
clf_gmm_plp = GaussianMixture(n_components=8, covariance_type='full',
max_iter=2000).fit(X_train_plp)

# Unsupervised Learning
# Predict labels for the test set
y_pred_plp = clf_gmm_plp.predict(X_test_plp)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test_plp, y_pred_plp)
print("Accuracy:", accuracy*100)
```

Predicted Accuracy of this model was 20.94%

Training SVM Classifier on MFCC Features

```
clf_svm_mfcc = svm.SVC(C=10, kernel='linear', gamma='scale').fit(X_train_mfcc,
y_train_mfcc)

# Predict labels for the test set
y_pred_mfcc = clf_svm_mfcc.predict(X_test_mfcc)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test_mfcc, y_pred_mfcc)
print("Accuracy:", accuracy*100)
```

Predicted Accuracy of this model was 84.78%. To achieve this accuracy, optimal hyper-parameters were found using *Grid Search*.

```
# Define the parameter grid for grid search
param_grid = {
    'C': [0.1, 1, 10, 100],          # Penalty Factor
    'kernel': ['linear', 'rbf'],     # Kernel type
    'gamma': ['scale', 'auto']       # Kernel coefficient}

# Create the SVM model
svm_model_mfcc = svm.SVC()

# Perform grid search using cross-validation
grid_search = GridSearchCV(svm_model_mfcc, param_grid, cv=5)
```

```
grid_search.fit(X_train_mfcc, y_train_mfcc)

# Print the best hyperparameters
print("Best Hyperparameters:", grid_search.best_params_)

# Evaluate the model with best hyperparameters on the test set
best_model = grid_search.best_estimator_
accuracy = best_model.score(X_test_mfcc, y_test_mfcc)
print("Accuracy with Best Hyperparameters:", accuracy*100)
```

Training SVM Classifier on PLP Features

```
clf_svm_plp = svm.SVC(C=100, kernel='linear', gamma='scale').fit(X_train_plp,
y_train_plp)

# Predict labels for the test set
y_pred_plp = clf_svm_plp.predict(X_test_plp)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test_plp, y_pred_plp)
print("Accuracy:", accuracy*100)
```

Predicted Accuracy of this model was 36.47%.

Real Time Speaker Identification

```
duration = 5  # Duration for recording in seconds

# Define the threshold for speaker identification
threshold = 0.6

# Load the dictionary mapping class labels to speaker identities
classes = {"ali":1,"ayesha":2,"mahnoor":3,"maryam":4,"usman_amin":5,"askari":6,
"usman_hassan":7,"huzaifa":8}

# Function to extract MFCC features from audio
def get_key(dictionary, value):
    for key, val in dictionary.items():
        if val == value:
            return key
    return None

def extract_mfcc(audio, sam_r):
```

```
    mfccs = np.mean(librosa.feature.mfcc(y=audio.flatten(), sr=sam_r, n_mfcc=5).T,
 axis = 0)
    mfcc_features = mfccs.reshape(1,-1)
    return mfcc_features

# Function to identify the speaker from MFCC features
def identify_speaker(features):
    predicted_label = clf_svm_mfcc.predict(features)[0]

    print('The predicted label is ' + get_key(classes,predicted_label))

    confidence = np.max(clf_svm_mfcc.decision_function(features))

    if confidence > threshold:
        speaker = get_key(classes,predicted_label)
    else:
        speaker = "Unknown"

    return speaker

# Record audio in real-time and perform speaker identification
def real_time_speaker_identification():

    true_speaker = input('Enter the id of the speaker to be identified: ')

    # Open an audio stream for recording
    stream = sd.InputStream(samplerate=sam_r, channels=1)

    print("Recording has been started. Start Speaking in the mic...")
    stream.start()
    audio_frames = []

    # Record audio for the specified duration
    for _ in range(int(duration * sam_r)):
        audio_frames.append(stream.read(1)[0])

    # Stop the audio stream
    stream.stop()
    stream.close()

    print("Recording stopped. Processing on the Input Audio")

    # Convert the audio frames to a numpy array
    audio = np.array(audio_frames)
```

```
    # Save the recorded audio to a WAV file (optional)
    write("C:\\Users\\ApriZon\\Desktop\\CEP\\recorded_audio.wav", sam_r, audio)

    # Extract MFCC features from the recorded audio
    mfcc_features = extract_mfcc(audio, sam_r)

    # Identify the speaker from the MFCC features
    speaker = identify_speaker(mfcc_features)

    if speaker == true_speaker:
        print('System has successfully recognized the speaker from the database')
    else:
        print('Failed to Identify')

# Run the real-time speaker identification
real_time_speaker_identification()
```

Results

The results show that the combination of MFCC and SVM achieves the highest accuracy of 83.52% on the test set, while the other combinations of PLP/MFCC with classifiers only manages to achieve accuracy of less than 75%. These results indicate that MFCC combined with SVM is a more effective feature extraction method for real-time speech recognition than other methods.

Feature Extraction Method	Classification Method	Accuracy
PLP	SVM	36.47%
	GMM-UBM	20.94%
MFCC	SVM	84.78%
	GMM-UBM	53.36%

Conclusion

In conclusion, this study compares the performance of two feature extraction methods, PLP and MFCC, in combination with two classification algorithms, SVM and GMM-UBM, for real-time speaker identification. The results show that the combination of MFCC and SVM achieves the highest accuracy on the test set, indicating that this combination is more effective for speech recognition than the other combinations tested. This study provides insights into the design of real-time speaker identification systems, and highlights the importance of selecting appropriate feature extraction methods and classification algorithms for achieving the highest accuracy.

References

scikit-learn. (n.d.). Documentation: Support Vector Machines (SVM). Retrieved from https://scikit-learn.org/stable/modules/svm.html

NumPy Developers. (2021). NumPy Documentation. Retrieved from https://numpy.org/doc/

SciPy community. (2021). SciPy Reference Guide. Retrieved from https://docs.scipy.org/doc/scipy/reference/

sounddevice. (n.d.). Documentation: sounddevice - Real-time Audio I/O. Retrieved from https://python-sounddevice.readthedocs.io/

librosa. (n.d.). Documentation: Librosa - Audio and Music Signal Analysis in Python. Retrieved from https://librosa.org/doc/main/

spafe. (n.d.). Documentation: Speech Feature Extraction. Retrieved from https://github.com/spafe-project/spafe