Development and Evaluation of a Palestinian Regional Accent Recognition System Using Acoustic Features

Shireen Hamdeh, 1200208. Riwaa Assi, 1201419. Dana Imam, 1200121
Department of Electrical and Computer Engineering,
Birzeit University

Abstract- The paralinguistic information in a speech signal includes clues to the geographical and social background of the speaker. This paper is concerned with developing and evaluating a simple acoustic Palestinian regional accent recognition system. Palestinian accents encompass various sub-accents such as Jerusalem, Nablus, Hebron, and Ramallah. The goal of this project is to create a system capable of recognizing the accent of a speaker from a short speech segment, specifically distinguishing between these four regional accents. To achieve this goal, we applied the state of the art techniques used in speaker and language identification, namely, Gaussian Mixture Model (GMM) with Mel-frequency cepstral coefficients (MFCC) for feature extraction. The best result (accuracy of 65%) was obtained with 16 Gaussians.

I. Introduction

The speech signal encompasses paralinguistic information in addition to its linguistic content, such as the speaker's gender, accent, language, emotional state, and age. Accent variation is a significant challenge for the performance of Automatic Speech Recognition (ASR) systems. Addressing accent variation within a language continues to attract significant research interest. Recognizing a speaker's accent before speech recognition can facilitate the adaptation of ASR model parameters for that accent, thereby improving recognition performance.

Languages such as Arabic, English, Spanish, and Chinese contain a variety of regional dialects that can differ significantly and may even be mutually unintelligible. Arabic, in particular, comprises a family of related dialects with limited vocabulary overlap but a high overlap in phoneme inventories. The challenge with Arabic dialects is amplified by the existence of multiple dialects within the same country.

Traditionally, Arabic dialects are divided into eastern and western groups. Eastern dialects include Levantine, Gulf, Iraqi, and Egyptian dialects, while western dialects, also known as Maghrebi, include Moroccan, Algerian, Tunisian, and Libyan dialects. The Levantine dialects encompass Syrian, Lebanese, Palestinian, and Jordanian sub-dialects. Each region within these countries has its own distinct accent, making it difficult for an Arabic ASR trained on Levantine dialects to perform well on Palestinian speech. Moreover, even within Palestinian dialectal speech, there are clear distinctions between accents from different regions, such as Hebron and Ramallah.

Previous work on Arabic dialects has mainly focused on broad categories such as Levantine, Gulf, Iraqi, Egyptian, and Maghrebi. However, in Palestine, the Bedouin accent is now rare, and there is a clear differentiation between the accents of city dwellers from different cities and those living in rural areas. This motivates a regional classification of Palestinian accents. To our knowledge, no prior work has addressed the task of regional accent recognition for Palestinian Arabic.

This paper addresses the problem of identifying individual Arabic accents among four regional accents: Jerusalem (JE), Hebron (HE), Nablus (NA), and rural Ramallah (RA). Our system achieved an accuracy of 65%.

The rest of this paper is structured as follows: the next section discusses related work. Following that, the materials and methods are presented, including a description of the dataset and an overview of the GMMs. This is followed by the presentation and discussion of experiments and results. Finally, conclusions and future directions are outlined.

II. Related Work

Over the last few years, much research has been conducted in the field of accent/dialect recognition. The most successful approaches applied for accent recognition can be divided into two major classes: phonotactic based approaches and acoustic based approaches [1-2]. Phonotactic approaches, such as Phone Recognition followed by Language model (PRLM) [4], use the difference in sequence of sounds for each particular accent for modeling accents, where acoustic approaches use the difference in realization (or pronunciation) of these sounds for building accent dependent models. The most common and successful acoustic approaches applied for accent recognition are those which use Gaussian Mixture Models (GMM) for building an accent independent model (called Universal Background Model, or UBM) and then use MAP adaptation for adapting UBM parameters for each target accent [5]. This system is known as GMM-UBM in the literature. [3]

Recent research has focused on the automatic recognition of regional Arabic Palestinian accents from four distinct regions: Jerusalem (JE), Hebron (HE), Nablus (NA), and Ramallah (RA). To achieve this, state-of-the-art techniques in speaker and language identification were applied, including GMM-UBM, Gaussian Mixture Model – Support Vector Machines (GMM-SVM), and the I-vector framework. These systems were trained and tested on speech data from 200 speakers. Both GMM-SVM and I-vector systems outperformed the baseline GMM-UBM system, with the I-vector system achieving the highest accuracy of 81.5% using 64 Gaussian components, compared to 73.4% accuracy by human listeners on the same test utterances. This demonstrates the potential of advanced modeling techniques to enhance accent recognition accuracy [3].

Recent advancements in accent recognition have also explored integrating visual features to improve system performance. Visual features, such as optical flow motion, have been combined with acoustic features to create robust multimodal accent recognition systems. Techniques like the Lucas-Kanade method and Horn-Schunck technique capture local and global motion features, respectively. When combined with Mel Frequency Cepstral Coefficients (MFCC), these features provide comprehensive data for classification. Deep learning approaches, including artificial neural networks and convolutional neural networks, have been employed to process these multimodal inputs, yielding high accuracy in accent recognition tasks. For instance, a multimodal system for Tamil accents achieved recognition rates of 93.7% using acoustic features, 89.5% with visual features, and 96% when combining both. These advancements highlight the potential of combining acoustic and visual data to improve the robustness and accuracy of accent recognition systems [6].

III. METHODOLOGY

This section outlines the methodology employed for the automatic recognition of regional Arabic Palestinian accents, focusing primarily on the use of Gaussian Mixture Models (GMMs) to classify the dataset provided for four distinct regional accents: Jerusalem (JE), Hebron (HE), Nablus (NA), and Ramallah (RA).

A. Feature Extraction using Mel Frequency Cepstral Coefficients (MFCC)

The MFCC feature extraction process, incorporating pre mphasis, windowing, spectral analysis, Mel filtering, cepstral analysis, and derivative computation, yields a compact representation of the speech signal that is suitable for various Automatic Speech Recognition (ASR) tasks.

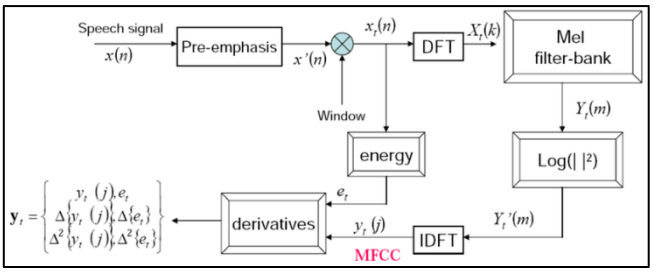

Figure 1: Mel Frequency Cepstral Coefficients Block Diagram [7]

As shown in Figure 1. The MFCC feature extraction process involves the following steps:

1. Pre-Emphasis Filter:

The pre-emphasis filter applies a first-order high-pass filter to the signal to compensate for the high-frequency attenuation. It is represented by the equation:

$$y(n) = s(n) - \alpha s(n-1)$$

2. Windowing:

Each frame of the pre-emphasized signal is multiplied by a window function $w(n)$ to minimize spectral leakage. The Hamming window, for instance, is given by:

$$f(x) = \begin{cases} 0.54 - 0.46\cos(\frac{2\pi n}{L-1}), & 0 < n < L-1 \\ 0 \quad , & otherwise \end{cases}$$

3. Discrete Fourier Transform (DFT):

The windowed signal is then transformed into the frequency domain using the Discrete Fourier Transform (DFT):

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{-j\frac{\pi}{N}kn}$$

4. Mel-Filter Bank:

Mel-scale is approximately linear below 1 kHz and logarithmic above 1 kHz

$$Mel(f) = 2595 log_{10}(1 + \frac{f}{700})$$

Then the output enters a square logarithmic block as follow:

$$\log |Mel(f)|^2$$

5. Discrete Cosine Transform (DCT):

The Inverse Discrete Fourier Transform (IDFT) doesn't produce a time domain output due to changes by the Mel filter and logarithmic scale, resulting in the cepstral domain. The Discrete Cosine Transform (DCT) simplifies the process, isolating vocal tract features by capturing low-order components and discarding the source signal. For a 24xN input, truncating the DCT output to the first 12xN components is essential for Speech-to-Text (STT) applications, minimizing individual voice differences. Additionally, DCT produces uncorrelated features, simplifying matrix operations and enhancing AI classification. These Mel-Frequency Cepstral Coefficients (MFCCs) are crucial for speech recognition.

$$y_t(k) = \sum_{m=1}^{M} \log(|Y_t(m)|) \cos\left(k(m-0.5)\frac{\pi}{M}\right), k = 0, \dots, J$$

6. Derivatives

To capture temporal dynamics, the first and second-order derivatives of the cepstral coefficients are computed:

$$d(t) = \frac{c(t+1) - c(t-1)}{2}$$

B. Noise Elimination

Noise elimination, especially in signal processing or data analysis, often involves normalizing the data to mitigate the effects of unwanted noise. A common method for normalizing data is to subtract the mean and divide by the variance. This process ensures that the data has a mean of zero and a variance of one, making it easier to identify and analyze the underlying signal.

Given a dataset $X = \{x_1, x_2, \ldots, x_n\}$, the mean μ and variance σ^2 are calculated as follows:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Given a dataset $X = \{x_1, x_2, \ldots, x_n\}$, the mean μ and variance σ^2 are calculated as follows:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu)^2$$

To normalize the data, each data point x_i is transformed using the formula:

$$z_i = \frac{x_i - \mu}{\sigma^2}$$

This transformation, known as standardization, results in a new dataset $Z = \{z_1, z_2, \ldots, z_n\}$, where each z_i represents the normalized value of the corresponding x_i. The process effectively eliminates noise by centering the data around zero and scaling it to unit variance, thereby highlighting the true structure of the signal and making further analysis more robust and reliable. This technique is particularly useful in various applications such as image processing, audio signal filtering, and machine learning preprocessing.

C. Gaussian Mixture Model (GMM)

A Gaussian Mixture Model (GMM) is a probabilistic model that assumes that all the data points are generated from a mixture of several Gaussian distributions with unknown parameters. Each Gaussian distribution represents a cluster in the dataset, and the GMM is parameterized by the mean, covariance, and weight of each Gaussian component.

1. Initialization of GMM using K-Means

To initialize the parameters of a GMM, K-Means clustering can be used. Here's how it works:

K-Means Clustering:

- **Number of Clusters**: Set the number of clusters KKK equal to the number of Gaussian components in the GMM.
- **Cluster Centroids:** Run K-Means to partition the data into KKK clusters. The centroids of these clusters will be used to initialize the means of the Gaussian components.
- **Cluster Assignment:** Assign each data point to the nearest centroid.

Initialization of GMM Parameters

- **Means (μ_m):** The centroids obtained from K-Means will be used to initialize the means of the Gaussian components.
- **Weights (π_m)**: The weight of each Gaussian component is initialized to the fraction of data points assigned to each cluster:

  $$\pi_m = \frac{N_m}{N}$$

  Where N_m is the number of points in cluster m and N is the total number of data points.
- **Covariances (Σ_m)**: The covariance matrix for each Gaussian component is initialized to the empirical covariance of the points assigned to the corresponding cluster:

2. GMM Model

Once initialized, the GMM parameters are refined using the Expectation-Maximization (EM) algorithm, which iteratively improves the parameter estimates.

Equations

1. Probability Density Function of a Multivariate Gaussian:

$$\Sigma_{\boldsymbol{m}} = \sum_{\boldsymbol{i \in Culsters_m}} (x_i - \mu_m)(x_i - \mu_m)^T$$

Where x is the data point, is the mean vector μ_m, Σ_m is the covariance matrix, and d is the dimensionality of the data.

Mixture Model

$$p(x) = \sum_{m=1}^{M} \pi_m \, N(x|\mu_m, \Sigma_{\boldsymbol{m}})$$

Where π_m is the weight of the $m - th$ Gaussian Component.

Expectation-Maximization Algorithm:

- **E-Step:** Calculate the probability that training data belongs to mixture component m

$$p\left(m|x_i, \theta^{(n)}\right) = \frac{\pi_m^{(n)} N\left(x_i^{(n)}|\mu_m^{(n)}, \sum_{\boldsymbol{m}}^{(\boldsymbol{n})}\right)}{\sum_{\boldsymbol{m=1}}^{\boldsymbol{M}} \pi_m^{(n)} N\left(x_i^{(n)}|\mu_m^{(n)}, \sum_{\boldsymbol{m}}^{(\boldsymbol{n})}\right)}$$

- **M-Step**: Update of each parameter is weighted by $w_{im} = p\left(m|x_i, \theta^{(n)}\right)$

$$\pi_m^{(n+1)} = \frac{1}{N}\sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} w_{im}$$

$$\mu_m^{(n+1)} = \frac{\sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} w_{im}\, x_i}{\sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} w_{im}}$$

$$\Sigma_k^{(n+1)} = \frac{\sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} (x_i - \mu_k)(x_i - \mu_k)^T}{\sum_{\boldsymbol{k=1}}^{\boldsymbol{K}} w_{im}}$$

IV. Experiments and Results

A. Experimental setup

One system was represented as a diagonal covariance GMM. It was trained on the whole training set of the four accents. The variance flooring was used in each iteration of EM algorithm during the system training.

To fasten the system, we use k-means as initialization of the GMM model such that the number of clusters is the same as the number of components.

In order to investigate the effect of number of Gaussian components on our system, we trained our system with different number of components (8, 16, 32, and 64).

B. Results and discussion

The experimental results of our system shown in Table 1, using the accuracy, Precision, Recall, and F_1_Score Metrics. According to the results we can notice that with 16 component of GMM, we achieved the highest accuracy result. Furthermore, notice that increasing the accuracy was firstly proportional to the increasing of number component, however, we can notice gradual decrement of accuracy after using more than 16 component for GMM. This might happen due to insufficient data while training the model, which causes an overfitting.

Table 1: Precision, Recall, F1_Score, and Accuracy Results

Number of GMM Component	Accents	Precision	*Recall*	*F1*	*Accuracy*
8	*Hebron*	*0.43*	*0.60*	*0.50*	*45%*
	Jerusalem	*0.33*	*0.40*	*0.63*	
	Nablus	*0.50*	*0.60*	*0.55*	
	Ramallah-Reef	*1.00*	*0.20*	*0.33*	
16	*Hebron*	*1.00*	*0.60*	*0.75*	*65%*
	Jerusalem	*0.57*	*0.80*	*0.67*	
	Nablus	*0.50*	*0.80*	*0.62*	
	Ramallah-Reef	*1.00*	*0.40*	*0.57*	
32	*Hebron*	*1.00*	*0.40*	*0.57*	*60%*
	Jerusalem	*0.50*	*0.80*	*0.62*	
	Nablus	*0.50*	*0.80*	*0.62*	
	Ramallah-Reef	*1.00*	*0.40*	*0.57*	
64	*Hebron*	*0.64*	*0.40*	*0.50*	*55%*
	Jerusalem	*0.50*	*0.80*	*0.62*	
	Nablus	*0.43*	*0.60*	*0.50*	
	Ramallah-Reef	*1.00*	*0.40*	*0.57*	

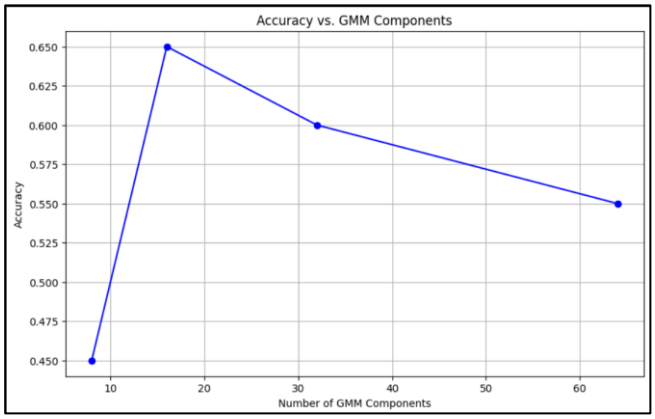

Figure 2: Effect of the GMM components on the system performance

V. Conclusion and Future Work

In conclusion, our study has successfully utilized advanced techniques, particularly Mel-Frequency Cepstral Coefficients (MFCC), and Gaussian Mixture Models (GMM) using K-means clustering as initialization, to accurately identify speaker accents within the realm of regional Arabic Palestinian dialects. The experimental results, as shown in Table 1, demonstrate that using 16 components in the GMM yields the highest accuracy. Initially, increasing the number of components improved accuracy, but beyond 16 components, a gradual decrease in accuracy was observed. This decrease is likely due to overfitting, caused by insufficient data during model training.

The findings validate our initial hypothesis regarding the effectiveness of GMM in accent identification. However, they also highlight the need for addressing data limitations. For future work, expanding the dataset by including more speakers and a wider variety of speech samples is crucial. This will enhance the system's performance, accuracy, and generalizability, paving the way for further advancements in speaker accent identification within this dialect group.

VI. Partners Participation Tasks

All of us worked equally on every aspect of this project. Together, we carefully studied the project requirements, discussed the methodology, and collaboratively developed the system's code. Each partner contributed to data collection, preprocessing, feature extraction, model training, evaluation, and documentation, ensuring a balanced and comprehensive approach to the project's development and execution.

References

[1] Hanani, A., Russell, M.J. and Carey, M.J. (2013) 'Human and computer recognition of regional accents and ethnic groups from British English speech', *Computer Speech & Language*, 27(1), pp. 59–74. doi:10.1016/j.csl.2012.01.003.

[2] Libraries, C.U. (1970) Automatic dialect and accent recognition and its application to speech recognition, Academic Commons. Available at: https://doi.org/10.7916/D8M61S68 (Accessed: 08 June 2024).

[3] Hanani, A. et al. (2015) 'Palestinian arabic regional accent recognition', 2015 International Conference on Speech Technology and Human-Computer Dialogue (SpeD) [Preprint]. doi:10.1109/sped.2015.7343088.

[4] Libraries, C.U. (1970) *Automatic dialect and accent recognition and its application to speech recognition*, *Academic Commons*. Available at: https://doi.org/10.7916/D8M61S68 (Accessed: 08 June 2024).

[5] Reynolds, D.A., Quatieri, T.F. and Dunn, R.B. (2000) 'Speaker verification using adapted gaussian mixture models', *Digital Signal Processing*, 10(1–3), pp. 19–41. doi:10.1006/dspr.1999.0361.

[6] N. Radha, Sachin Madhavan R M. (2021) 'Acoustic-Visual based Accent Identification System using Deep Neural Networks'. doi:10.24205/03276716.2020.4093.

[7] I. MEKKI. (2020), 'Automatic Speech Recognition in the French Language'.doi:10.13140/RG.2.2.12903.06564.

APPENDIX

```
import os
import numpy as np
import librosa
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report,
accuracy_score
from sklearn.mixture import GaussianMixture
import joblib  # For saving intermediate results

# Define the folder paths
baseFolder = 'drive/MyDrive/Dataset/Training'
testingBaseFolder = 'drive/MyDrive/Dataset/Testing'
regions = ['Hebron', 'Jerusalem', 'Nablus',
'Ramallah_Reef']

# Function to process a batch of audio files
def process_batch(files, regionFolder, maxNumFrames):
    batchFeatures = []
    batchLabels = []
    for wavFile in files:
        # Load the audio file
        filePath = os.path.join(regionFolder, wavFile)
        audio, fs = librosa.load(filePath, sr=None)

        # Extract MFCC features
        features = librosa.feature.mfcc(y=audio, sr=fs)

        # Normalize the MFCC features
        mean = np.mean(features, axis=1, keepdims=True)
        variance = np.var(features, axis=1,
keepdims=True)
        features = (features - mean) / variance

        # Update the maximum number of frames
        maxNumFrames = max(maxNumFrames,
features.shape[1])

        # Store the features and the label (region)
        batchFeatures.append(features)
        batchLabels.append(regionFolder.split('/')[-1])
    return batchFeatures, batchLabels, maxNumFrames

# Initialize lists to store the features and labels
allFeatures = []
allLabels = []

# Variable to track the maximum number of frames
maxNumFrames = 0

# Process the data in batches
batch_size = 10
for r in regions:
    regionFolder = os.path.join(baseFolder, r)
    wavFiles = [f for f in os.listdir(regionFolder) if
f.endswith('.wav')]

    print(f'Processing folder: {r}')

    for i in range(0, len(wavFiles), batch_size):
        batchFiles = wavFiles[i:i + batch size]
        batchFeatures, batchLabels, maxNumFrames =
process batch(batchFiles, regionFolder, maxNumFrames)

        allFeatures.extend(batchFeatures)
        allLabels.extend(batchLabels)

        # Save intermediate results
        joblib.dump((allFeatures, allLabels,
maxNumFrames), f'{r} intermediate results.pkl')

        print(f'Processed batch {i//batch_size + 1} of
{len(wavFiles)//batch size + 1}')

# Pad or trim features and concatenate them
featuresMatrix = np.zeros((allFeatures[0].shape[0],
maxNumFrames, len(allFeatures)))
for i, feature in enumerate(allFeatures):
```

```
    numFrames = feature.shape[1]
    if numFrames < maxNumFrames:
        # Pad features with zeros
        padding = np.zeros((feature.shape[0], maxNumFrames - numFrames))
        feature = np.hstack((feature, padding))
    elif numFrames > maxNumFrames:
        # Trim features
        feature = feature[:, :maxNumFrames]
    featuresMatrix[:,:,i] = feature

# Convert labels to categorical
labels = np.array(allLabels)

# Train GMM for each region using K-means
numComponents = 16
gmmModels = []

for r in regions:
    print(f'Training GMM for region: {r}')

    # Extract features for the current region
    regionFeatures = featuresMatrix[:,:,labels == r]

    # Convert the features to a 2D matrix where each row is a feature vector
    regionFeatures2D = regionFeatures.reshape(regionFeatures.shape[0], -1).T

    # Check if there are enough data points for GMM training
    if regionFeatures2D.shape[0] < numComponents:
        print(f'Error: Insufficient data for region {r}')
        continue  # Skip training for this region

    # Apply K-means clustering
    kmeans = KMeans(n_clusters=numComponents, random_state=0, n_init=10).fit(regionFeatures2D)
    labels kmeans = kmeans.labels
    centroids = kmeans.cluster centers

    # Calculate weights and covariances
    weights = []
    covariances = []
    for k in range(numComponents):
        points_in_cluster = regionFeatures2D[labels_kmeans == k]
        weights.append(len(points_in_cluster) / len(regionFeatures2D))
        cov matrix = np.cov(points in cluster, rowvar=False) + 1e-6 * np.eye(points_in_cluster.shape[1])
        covariances.append(np.diag(cov_matrix))  # Use the diagonal of the covariance matrix

    # Train GMM using centroids, weights, and covariances
    gmm = GaussianMixture(n_components=numComponents, covariance_type='diag')
    gmm.weights  = np.array(weights)
    gmm.means_ = centroids
    gmm.covariances_ = np.array(covariances)
    gmm.precisions_cholesky_ = 1 / np.sqrt(gmm.covariances )  # For diagonal covariances, precision is 1/variance

    gmmModels.append(gmm)

    print(f'Finished training GMM for region: {r}')

# Testing the trained GMM models
predictedLabels = []
trueLabels = []

# Loop over each region in the testing dataset
for r in regions:
    regionFolder = os.path.join(testingBaseFolder, r)
    wavFiles = [f for f in os.listdir(regionFolder) if f.endswith('.wav')]

    print(f'Testing region: {r}')

    # Iterate through each audio file in the region folder
    for wavFile in wavFiles:
        # Load the audio file
        filePath = os.path.join(regionFolder, wavFile)
        audio, fs = librosa.load(filePath, sr=None)

        # Extract MFCC features
        features = librosa.feature.mfcc(y=audio, sr=fs)

        # Normalize the MFCC features
        mean = np.mean(features, axis=1, keepdims=True)
        variance = np.var(features, axis=1, keepdims=True)
        features = (features - mean) / variance
```

```
        # Calculate log likelihood for each GMM model
        logLikelihoods = np.zeros(len(gmmModels))
        for i, gmm in enumerate(gmmModels):
            # Calculate log likelihood
            logLikelihoods[i] =
np.sum(gmm.score(features.T))

        # Find the region with the maximum log likelihood
        predictedRegionIdx = np.argmax(logLikelihoods)

        # Store predicted and true labels
        predictedLabels.append(regions[predictedRegionIdx
])
        trueLabels.append(r)

        print(f'Predicted region for file {wavFile}:
{regions[predictedRegionIdx]}')

# Calculate accuracy
correctPredictions = np.sum(np.array(predictedLabels) ==
np.array(trueLabels))
totalFiles = len(predictedLabels)
accuracy = correctPredictions / totalFiles * 100

print(f'Accuracy: {accuracy:.2f}%')

# Convert trueLabels and predictedLabels to numpy arrays
trueLabels = np.array(trueLabels)
predictedLabels = np.array(predictedLabels)

# Calculate accuracy
accuracy = accuracy score(trueLabels, predictedLabels)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Generate and print the classification report
report = classification_report(trueLabels,
predictedLabels, target_names=regions)
print(report)
```