Simple Acoustic Palestinian Regional Accent Recognition System

Fayez Backleh - 1190216, Pierre Backleh - 1201296, Dana Shtieh 1191868

Faculty of Engineering and Technology
Department of Electrical and Computer Engineering

Birzeit University, Ramallah, Palestine

***Abstract*—This paper presents the development of an accent recognition system for the native speakers of Palestine using machine learning. The results of the work serve as baseline for the advancement of recognizing Palestinian speakers with Jerusalem, Hebron, Ramallah, and Nablus accents in Arabic language. A set of audio files is prepared as script for the development of the speech corpus. The script is used to capture the four Palestinian accents mentioned above. The corpus was validated, cleaned and divided into training and testing. Afterwards, librosa library is utilized to analyze and extract prosodic features such as MFCCs, energy, and zero-crossing rate. The model was tested and yields 70% and 75% accuracy for MLP and KNN models respectively.**

Keywords-machine learning, Palestinian accent, Arabic Language, recognition and classification, MFCC, ZCR, KNN, MLP.

I. INTRODUCTION

Speech recognition technology can enhance human–computer interaction and has significant applications in various fields such as smart homes and medical rehabilitation [1]. In actual speech, the presence of accents can significantly impact recognition performance [2]. Accent can be defined as the pronunciation style in a language. In the Palestinian setup, the accent of a speaker can be highly influenced by other speakers in an approximate geographical location. For this reason, people can speak the same language but with a different accent resulting to a language used with multiple accents [3].

Automatic accent recognition, also known as accent identification, is based on the consistency of acoustic patterns that can be identified in speaking style that leads to the distinction of pronunciation on the same accent cluster [3]. There have been numerous research efforts to develop accent recognition system in different languages. However, as of the writing of this paper, the work is one of the existing studies that explored accent recognition in Palestinian languages.

II. RELATED WORKS

There are different efforts made to develop accent recognition in different regions using various approaches. Various models created from acoustic features, but also deep neural networks were explored.

Gaikwad (2013) investigated the English pronunciation of native Arabic speakers, focusing on acoustic features such as energy, pitch, and formant frequency. Their findings indicated that the energy feature was particularly effective in identifying the accents of Arabic speakers [4].

Pham (2016) combined Mel Frequency Cepstrum and F0 in a Gaussian Mixture Model to recognize Vietnamese dialects. They found that integrating formants and bandwidths with normalized F0 improved the baseline dialect identification accuracy from 58.6% to 72.2% [5].

Biadsy (2011) explored various methods utilizing different acoustic features of speech signals, including frame-based acoustic features, phonetic and phonotactic features, and high-level prosodic features, to develop a system capable of recognizing regional dialects and accents. The most successful method in this study was tested on four broad Arabic dialects, ten Arabic subdialects, American vs. Indian English accents, American Southern vs. Non-Southern dialects, state-level American dialects plus Canada, and three Portuguese dialects. This approach achieved an Equal Error Rate (EER) of 4% for broad Arabic dialects, 6.3% for American vs. Indian English accents, 14.6% for American Southern vs. Non-Southern dialects, and 7.9% for Portuguese dialects. Additionally, applying this method to an automatic speech recognition system resulted in a significant improvement of 4.6% [6].

Hanani (2015) explores the recognition of regional Arabic Palestinian accents using techniques like GMM-UBM, GMM-SVM, and I-vectors, with the I-vector system showing superior performance. Human listeners achieved an average accuracy of 73.4% in identifying accents [7].

Additionally, various studies have been conducted on speaker and language recognition, employing techniques like support vector machines, Gaussian mixture models, deep learning, and i-vectors to enhance accuracy and efficiency.

III. BACKGROUND

This section will discuss the theoretical background of the extracted features and the machine learning models utilized in this project.

The models employed in our project are as follows:

A. *K-nearest neighbors (KNN)*

The k-nearest neighbors (KNN) algorithm is a supervised, non-parametric learning classifier used to predict or classify data points based on their proximity to one another. Despite its primary application in classification, KNN is also widely regarded as one of the simplest and most popular classifiers in machine learning for both classification and regression tasks. The core principle of KNN is that similar data points tend to be near each other, allowing the algorithm to make predictions based on the closeness of these points [8].

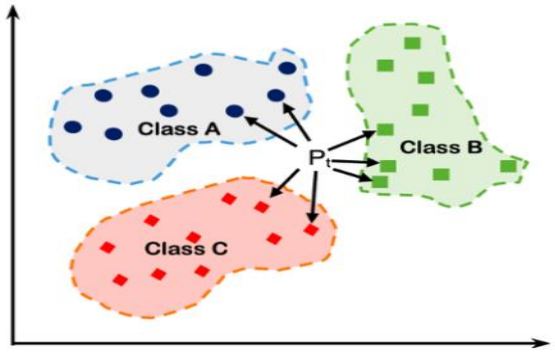

Figure 1: K-nearest neighbors Diagram.

B. *Multilayer Perceptron (MLP)*

The multilayer perceptron is a commonly used neural network [24–26]. MLP is composed of multiple layers, including an input layer, hidden layers, and an output layer, where each layer contains a set of perception elements known as neurons, and to create a neural network, neurons are combined together so that the outputs of some neurons are inputs of other neurons.

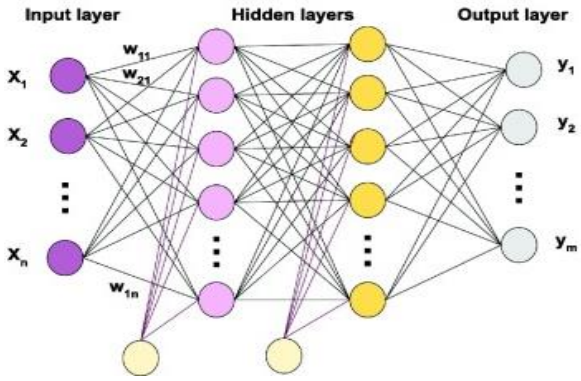

Figure 2: Multilayer Perceptron Diagram.

The features that are extracted from our audio files are the follows:

A. *Mel-frequency cepstral coefficients (MFCCs)*

Mel-frequency cepstral coefficients (MFCCs) are a representation of the spectral envelope of a sound signal, which are commonly used in speech recognition systems. They are calculated by taking the Fourier transform of the sound signal, mapping the resulting spectrum onto the mel scale, taking the logarithm of the powers at each mel frequency, applying the discrete cosine transform (DCT), and keeping a subset of the resulting coefficients. The number of coefficients kept is typically between 10 and 20, and they represent the most significant features of the sound signal. MFCCs are calculated using a series of mathematical operations, including the Fourier transform, mel-scale mapping, logarithmic scaling, and DCT [10].

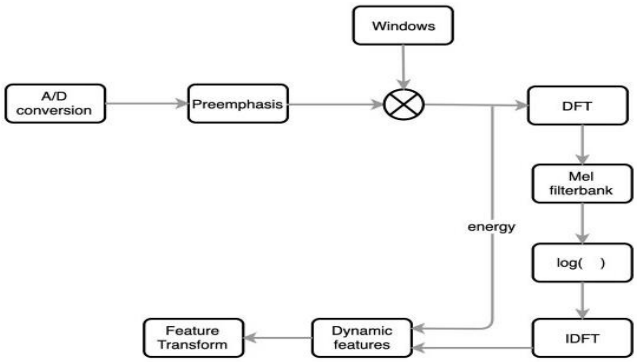

Figure 3: Mel-frequency cepstral coefficients Block Diagram

B. *Zero Crossing Rate (ZCR)*

The Zero-Crossing Rate (ZCR) gauges how quickly a signal crosses the zero-amplitude axis, moving from positive to negative or from negative to positive. In the field of sound recognition, this measure is very important, especially for speech recognition and music retrieval. Because voiced sounds are periodic, they usually have a lower ZCR than unvoiced sounds, which aids in the distinction between voiced and unvoiced speech segments in speech recognition. Due to their transitory, high-frequency content, percussion sounds typically have higher ZCR values, making ZCR an important classification tool for music information retrieval. As a result, ZCR is an essential component of many algorithms for sound classification, improving the precision and effectiveness of identifying and classifying diverse audio signals [11].

C. *Energy:*

It measures the intensity of the speech signal and helps differentiate between accents by capturing variations in speech loudness and emphasis. By analyzing the energy levels in different parts of the speech signal, it is possible to identify patterns and characteristics specific to certain accents, making it a valuable tool in accent identification and classification tasks.

IV. METHODOLOGY

1. *Data Collection*

The first step in training a machine learning model to identify various regional accents using wav files is to arrange the dataset, which consists of 40 wav files (10 from each of the four regions), into an appropriately labeled organized directory. The audio recordings should then be preprocessed to extract pertinent features like Mel-Frequency Cepstral Coefficients (MFCCs).

A distinct testing dataset comprising 20 wav files is used to assess the efficacy of the trained accent identification model (5 from each of the four areas). We first properly label and arrange these files. Lastly, we evaluate the model's ability in identifying the regional accents in the testing data by calculating measures such as accuracy.

2. *Features Selection*

Various feature combinations were tested to identify the most effective set for training the accent recognition model. MFCCs, Energy and Zero-crossing Rate features.

Mel-frequency cepstral coefficients are coefficients that are frequently employed in audio and speech processing. They describe the short-term power spectrum of a sound. There are 12 MFCC (Mel Frequency Cepstral Coefficients) coefficients, according to our calculations.

Energy, which is the audio signal's root mean square (RMS) energy.

Zero-Crossing Rate: The rate at which a signal switches signs; this information might be helpful in differentiating between various audio signal types.

3. *Modelling*

The project offers a Multi-layer Perceptron (MLP) neural network and K-Nearest Neighbors (KNN) as two options for training models. Users can choose files, evaluate the results, and choose between two classification approaches (KNN and MLP) via the GUI.

K-Nearest Neighbors (KNN):

For audio-based accent recognition with K-Nearest Neighbors (KNN) classification. First, it uses librosa to extract audio attributes including energy, zero-crossing rate, and MFCCs. Standard Scaler is used for preprocessing and scaling these features. A KNN model is initialized and trained by the NN class using the given number of neighbors (K=6) which was determined to be the best choice through experimentation.

The .wav files in the training directory provide the training data, which is processed and consists of features and encoded labels. The model predicts the accent after processing the chosen file, extracting and scaling its features.

Multi-layer Perceptron (MLP):

As mentioned in the code, The train_mlp_model function, which initializes an MLP classifier with two hidden layers containing 50 and 30 neurons, the “relu” activation function, and the Adam optimizer, is used to create the Multi-Layer Perceptron (MLP) capability. Using the features and labels that were collected from the training data, the MLP model is trained.

V. EXPERIMENTS AND RESULTS

First, all the audio files were tested using the MLP and KNN models respectively and the results of accuracy, precision, recall and F1 score are shown in the two figures below.

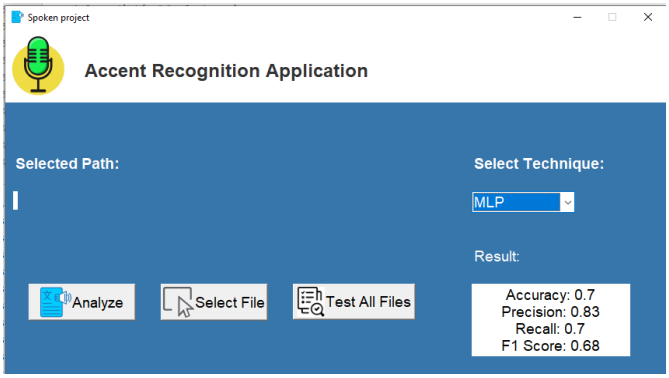

Figure 4: accuracy, precision, recall and F1 score results for MLP Model

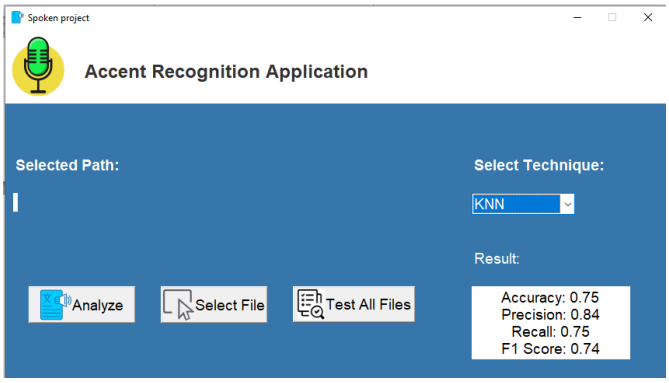

Figure 5: accuracy, precision, recall and F1 score results for KNN Model.

Now, a random checking for the audio files were assigned in both methods, some of the predictions are correct and the others has false prediction. Figures below show some of the tests that were applied to the testing audio files list:

The figure below shows a correct prediction for Ramallah Reef accent using KNN model.

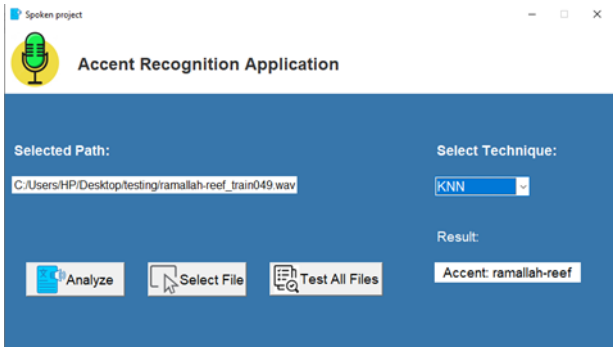

Figure 6: Correct Prediction for Ramallah Accent Using KNN model.

In the figure below is another correct prediction for Jerusalem accent using the MLP model

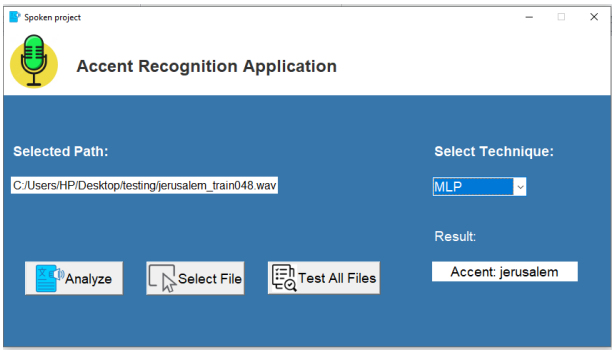

Figure 7: Correct Prediction for Jerusalem Accent Using MLP model.

The figure below has a false prediction case for Nablus accent using MLP model

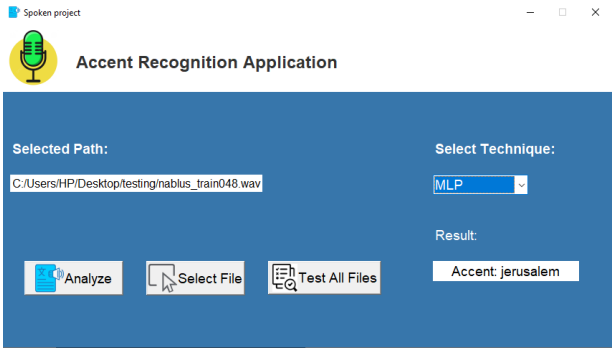

Figure 8: False Prediction for Nablus Accent Using MLP model.

The MLP classifier incorrectly predicted the Nablus accent as Jerusalem. This misclassification could be due to the similarities in acoustic features between these two accents or insufficient training data for Nablus.

Finally, another false prediction for Hebron accent using KNN model is shown in the figure below.

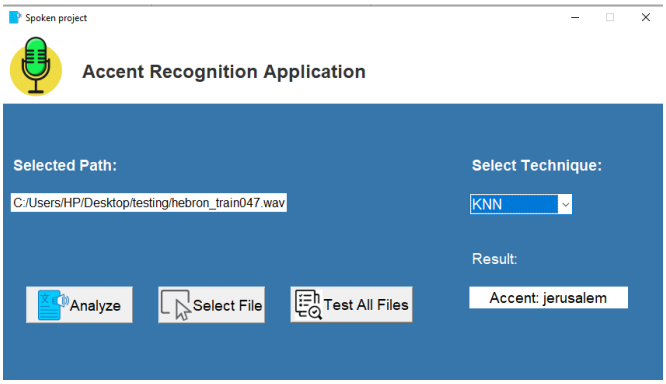

Figure 9: False Prediction for Hebron Accent Using KNN model.

The KNN classifier made a similar error by predicting the Hebron accent as Jerusalem. This further suggests that the Jerusalem accent's features might overlap significantly with those of other accents, leading to confusion.

Depends on the results shown in this section, for many real-world applications, MLPs (and neural networks in general) tend to outperform simpler models like KNN in terms of accuracy and ability to generalize, especially with large datasets and complex tasks. However, KNN remains valuable for its simplicity and ease of use in appropriate contexts. So, since we don't have a massive training data, KNN gives better results than MLP in terms of accuracy, precision, recall and F1 score.
The classification of performance for the four accents is shown by the confusion matrix for kNN model as follows:

Table 1 - Confusion Matrix For KNN

Actual \ Predicted	Nablus	Hebron	Ramallah	Jerusalem
Nablus	2	0	0	3
Hebron	0	4	0	1
Ramallah	0	0	4	1
Jerusalem	0	0	0	5

This confusion matrix shows that while the model works well for certain accents, it sometimes confuses the accents of Ramallah and Nablus with that of Jerusalem.

VI. CONCLUSION AND FUTURE WORK

In this project, we have effectively created an accent identification system that is intended to recognize regional Palestinian accents, such as those from Jerusalem, Hebron, Ramallah, and Nablus. The model produced encouraging results by building a comprehensive speech corpus and then doing a thorough analysis with the librosa package to extract important prosodic properties as MFCCs, energy, and zero-crossing rate. The machine learning approach for

regional accent recognition was validated by the system's reasonable accuracy. These results provide a baseline against which future work in the identification and classification of Palestinian accents will be measured.

As a future work, the speech model will be expanded to include more types of speakers and sub-dialects. Also, test it in real-world settings and investigate its potential applications to other regional dialects within the Arabic-speaking community, in order to verify the system's dependability and efficacy. These actions are intended to increase the system's accuracy and range of applications.

VII. Partners Participation Tasks

The entire team collaborated on the project, with Pierre focusing on the KNN and MLP functions, feature extraction, and GUI development. Fayez concentrated on evaluation, processing training data, and processing testing data. Dana handled reading .wav files, data preprocessing, and report writing.

VIII. References

[1]: Jat, Dharm Singh, Anton S. Limbo, and Charu Singh. "Speech-based automation system for the patient in orthopedic trauma ward." Smart biosensors in medical care. Academic Press, 2020. 201-214.

[2]: Berjon, Pierre, Avishek Nag, and Soumyabrata Dev. "Analysis of French phonetic idiosyncrasies for accent recognition." Soft Computing Letters 3 (2021): 100018.

[3]:chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://core.ac.uk/download/pdf/286965274.pdf

[4]: Gaikwad, Santosh, Bharti Gawali, and K. V. Kale. "Accent recognition for indian english using acoustic feature approach." International Journal of Computer Applications 63.7 (2013).

[5]: Hung, Pham Ngoc, Trinh Van Loan, and Nguyen Hong Quang. "Automatic identification of vietnamese dialects." Journal of Computer Science and Cybernetics 32.1 (2016): 19-30.

[6]: Biadsy, Fadi. Automatic dialect and accent recognition and its application to speech recognition. Columbia University, 2011.

[7]: Hanani, Abualsoud, et al. "Palestinian Arabic regional accent recognition." 2015 International Conference on Speech Technology and Human-Computer Dialogue (SpeD). IEEE, 2015.

[8]: https://www.ibm.com/topics/knn

[9]:https://www.sciencedirect.com/topics/computer-science/multilayer-perceptron#:~:text=The%20multi%2Dlayer%20perceptron%20(MLP,suitable%20for%20non%2Dlinear%20cases.

[10]:https://www.sciencedirect.com/topics/computer-science/cepstral-coefficient#:~:text=Mel%2Dfrequency%20cepstral%20coefficients%20(MFCCs)%20represent%20timbral%20information%20of,DCT%20to%20remove%20redundant%20information.

[11]:https://www.analyticsvidhya.com/blog/2022/01/analysis-of-zero-crossing-rates-of-different-music-genre-tracks/

IX. Appendix

```
#Fayez Backleh - 1190216
# Pierre Backleh - 1201296
# Dana Shtieh 1191868

# Import libraries
import os
import glob
import librosa
import numpy as np
from tkinter import *
from tkinter import ttk, filedialog
from sklearn.neighbors import
KNeighborsClassifier
from sklearn.neural_network import
MLPClassifier
from sklearn.preprocessing import
StandardScaler, LabelEncoder
from sklearn.metrics import
accuracy_score, precision_score,
recall_score, f1_score,
confusion_matrix
import warnings

warnings.filterwarnings('ignore')

# Directories for training and
testing data
training_data_dir =
r'C:\Users\HP\Desktop\Training'
testing_data_dir =
r'C:\Users\HP\Desktop\Testing'

# K value for KNN classifier
K = 6

# Setup GUI window
root = Tk()
root.title("Spoken project")
root.geometry("850x450+220+120")
root.resizable(False, False)
root.configure(bg="#3776ab")

selected_file_path = StringVar()
```

```
# Neural Network (NN) class for KNN classification
class NN:
    def __init__(self, trainingFeatures, trainingLabels) -> None:
        self.trainingFeatures = trainingFeatures
        self.trainingLabels = trainingLabels
        self.classifier = KNeighborsClassifier(n_neighbors=K)

self.classifier.fit(self.trainingFeatures, self.trainingLabels)

    def predict(self, features):
        return self.classifier.predict(features)

# Function to extract audio features from a file
def extract_features(audio_file_path):
    print(f"Extracting features from {audio_file_path}")
    y, sr = librosa.load(audio_file_path, sr=None, mono=True)
    mfccs = np.mean(librosa.feature.mfcc(y=y, sr=sr, n_mfcc=12), axis=1)
    energy = np.mean(librosa.feature.rms(y=y), axis=1)
    zero_crossing_rate = np.mean(librosa.feature.zero_crossing_rate(y=y), axis=1)

    # Combine all features into a single vector
    features = np.concatenate((mfccs, energy, zero_crossing_rate))
    return features

# Function to evaluate model predictions
def evaluate(labels, predictions):
    accuracy = accuracy_score(labels, predictions)
    precision = precision_score(labels, predictions, average='macro')
    recall = recall_score(labels, predictions, average='macro')
    f1 = f1_score(labels, predictions, average='macro')
    # Print the confusion matrix
    cmatrix = confusion_matrix(labels, predictions)
    return accuracy, precision, recall, f1, cmatrix

# Function to preprocess features using StandardScaler
def preprocess(features):
    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(features)
    return scaled_features, scaler

# Function to process training data from directory
def process_training_data(data_dir):
    features = []
    labels = []

    for filename in glob.glob(os.path.join(data_dir, '*.wav')):
        # Extract features
        extracted_features = extract_features(filename)

features.append(extracted_features)
        # Extract label from filename
        label = os.path.basename(filename).split('_')[0]
        labels.append(label)

    # Encode labels
    label_encoder = LabelEncoder()
    labels = label_encoder.fit_transform(labels)

    # Print encoded labels
    print(f"Encoded labels: {labels}")

    features = np.array(features)
    features, scaler = preprocess(features)

    return features, labels, scaler, label_encoder

# Function to preprocess testing data using the trained scaler
def preprocess_testing_data(data_dir, scaler):
    features = []
    labels = []
    file_paths = []

    for filename in glob.glob(os.path.join(data_dir, '*.wav')):
        extracted_features = extract_features(filename)
        scaled_features = scaler.transform([extracted_features]
```

```
)

features.append(scaled_features[0])

        label =
os.path.basename(filename).split('_')[
0]
        labels.append(label)
        file_paths.append(filename)

    return np.array(features), labels,
file_paths

# Function to process individual
testing audio file
def
process_testing_data(audio_file_path,
scaler):
    # Extract features
    extracted_features =
extract_features(audio_file_path)
    extracted_features =
scaler.transform([extracted_features])
    return
np.array(extracted_features)

# Function to train MLP model
def train_mlp_model(features,
results):
    model =
MLPClassifier(hidden_layer_sizes=(50,
30), activation='relu', max_iter=1000,
learning_rate_init=0.001,

solver='adam', verbose=True)
    model.fit(features, results)
    return model

# Function to select an audio file
def select_file():
    file_path =
filedialog.askopenfilename(filetypes=[
("WAV files", "*.wav")])
    selected_file_path.set(file_path)
    print(f"Selected file:
{file_path}")

file_path_label.config(text=file_path)

# Function to analyze the selected
file using the chosen model
def analyze():
    try:
        selected_technique =
technique_combobox.get()
        if selected_technique ==
"KNN":
            print("knn")
            model_nn = NN(X_train,
y_train)
            model =
model_nn.classifier
        elif selected_technique ==
"MLP":
            print("mlp")
            model =
train_mlp_model(X_train, y_train)
        else:
            print("Invalid
selection")

result_label.config(text="Invalid
selection")
            return

        test_file_path =
selected_file_path.get()

        if not
os.path.isfile(test_file_path):
            print("Invalid file path.
Please try again.")

result_label.config(text="Invalid
file path. Please try again.")
            return

        X_test =
process_testing_data(test_file_path,
scaler)

        predictions =
model.predict(X_test)
        predicted_label =
label encoder.inverse transform(predi
ctions)

        print(f"Predicted Accent:
{predicted_label[0]}")

result_label.config(text=f"Accent:
{predicted_label[0]}")
    except Exception as e:
        print(f"Error during
analysis: {e}")

result_label.config(text=f"Error:
{e}")

# Function to test all files
def test_all_files():
    try:
        selected_technique =
technique_combobox.get()
        if selected_technique ==
"KNN":
            print("knn")
            model_nn = NN(X_train,
y_train)
            model =
model_nn.classifier
        elif selected_technique ==
```

```
"MLP":
            print("mlp")
            model =
train_mlp_model(X_train, y_train)
        else:
            print("Invalid selection")

result_label.config(text="Invalid
selection")
            return

        if not test_file_paths:
            print("No .wav files found
in the test directory.")

result_label.config(text="No .wav
files found in the test directory.")
            return

        y_true = []
        y_pred = []

        for i, file in
enumerate(test_file_paths):
            # Extract true label from
filename
            true_label =
y_test_labels[i]
            true_label_encoded =
label_encoder.transform([true_label])[
0]

y_true.append(true_label_encoded)

            # Use preprocessed test
features
            X_test =
np.array([X_test_all[i]])

            # Make predictions
            predictions =
model.predict(X_test)

y_pred.append(predictions[0])

        # Evaluate the model
        accuracy, precision, recall,
f1, cmatrix = evaluate(y_true, y_pred)
        print(f"Accuracy: {accuracy}")
        print(f"Precision:
{precision}")
        print(f"Recall: {recall}")
        print(f"F1 Score: {f1}")
        print(f"Confusion
Matrix:\n{cmatrix}")

result_label.config(text=f"Accuracy:
{round(accuracy, 2)}\nPrecision:
{round(precision, 2)}\nRecall:
{round(recall, 2)}\nF1 Score:
{round(f1, 2)}")
    except Exception as e:
        print(f"Error during testing
all files: {e}")

result_label.config(text=f"Error:
{e}")


# Setup GUI for our application
iconImg1 =
PhotoImage(file="mainLogo2.png")
root.iconphoto(False, iconImg1)

# header
header = Frame(root, bg="white",
width=850, height=95)
header.place(x=0, y=0)

Logo = PhotoImage(file="logo1.png")
Label(header, image=Logo,
bg="white").place(x=6, y=7)
Label(header, text="Accent
Recognition Application",
font="Helvetica 18 bold", bg="white",
fg="#333", padx=10, pady=10).place(
    x=90, y=27)

pathLabel = Label(root,
text="Selected Path:", font="arial 14
bold", bg="#3776ab", fg="white")
pathLabel.place(x=10, y=160)

file_path_label = Label(root,
text="", font="arial 12", bg="white",
fg="black", wraplength=395)
file_path_label.place(x=10, y=210)

icon2 =
PhotoImage(file="mainLogo2.png")
button1 = Button(root,
text="Analyze", compound=LEFT,
image=icon2, width=130, font="arial
14", command=analyze)
button1.place(x=30, y=330)

iconSelect = PhotoImage(file="select-
file.png")
button2 = Button(root, text="Select
File", compound=LEFT,
image=iconSelect, width=130,
font="arial 14",
                 command=select_file)
button2.place(x=200, y=330)

iconTest = PhotoImage(file="test-
file.png")
button3 = Button(root, text="Test All
Files", compound=LEFT,
image=iconTest, width=150,
font="arial 14",
```

```
command=test_all_files)
button3.place(x=370, y=330)

comboLabel = Label(root, text="Select
Technique:", font="arial 14 bold",
bg="#3776ab", fg="white")
comboLabel.place(x=600, y=160)

# Combobox for selecting machine
learning technique
technique_combobox =
ttk.Combobox(root, values=["KNN",
"MLP"], font="arial 14",
state="readonly", width=10)
technique_combobox.place(x=600, y=210)
technique_combobox.current(0)

result_here = Label(root,
text="Result:", font="arial 14",
bg="#3776ab", fg="white")
result_here.place(x=600, y=280)

# Label to show the results
result_label = Label(root, text="",
font="arial 14", bg="white",
fg="black", width=18)
result_label.place(x=600, y=330)

print("Analyzing...")

# Process training data
print("Training files:")
X_train, y_train, scaler,
label encoder =
process_training_data(training_data_di
r)

# Preprocess testing data
print("testing files: ")
X_test_all, y_test_labels,
test_file_paths =
preprocess_testing_data(testing_data_d
ir, scaler)

root.mainloop()
```