Acoustic Palestinian regional accent recognition system

Amal Ziad 1192141 & Bara' Hamad 1191495
Department of Electrical and Computer Engineering
Birzeit University
Birzeit, Palestine
1192141@student.birzeit.edu & 1191495@student.birzeit.edu

***Abstract*— The project's aim is to develop and evaluate a straightforward acoustic recognition system to identify regional Palestinian accents. The target accents are as follows: Jerusalem, Nablus, Hebron, and Ramallah, each representing distinct linguistic characteristics. The system will analyze short speech segments and classify them into one of the four specified accents. The system aims to accurately understand and categorize the regional accents by using beneficial acoustic features extracted from the speech samples. This approach highlights the unique acoustic signatures associated with each accent, which supports accurate recognition and classification. The outcome of this project will contribute to the field of speech processing and dialectology, providing insights into the acoustic properties of Palestinian accents and their identification through automated systems. The experiments included the following training models: random forest classifier, SVM, and CNN. The output of the experiments has showed that random forest produced more accurate performance with 75% accuracy.**

Keywords— accent recognition, Palestinian accents, speech processing, speech classification, machine learning.

I. INTRODUCTION

Even though automatic speech recognition (ASR) systems have advanced significantly in recent years, they still frequently face challenges from the variability brought about by various dialects and accents. This is especially important in areas with a lot of linguistic diversity, like Palestine, where different regional accents are spoken. Accurately identifying these accents is essential for boosting ASR system performance as well as for a number of other uses, including linguistic research, language learning aids, and improving speech-based user interfaces.

The objective of this project is to create an acoustic accent recognition system that can identify speech segments and categorize them into one of four regional accents spoken in Palestine: Ramallah, Jerusalem, Nablus, and Hebron. It is possible to classify these accents using their distinctive phonetic and acoustic traits. Our system aims to enhance the robustness and adaptability of ASR systems in Palestinian Arabic by concentrating on these regional variations.[1]

Our method is taking speech signals and using machine learning techniques to separate the acoustic features into the appropriate accents. Mel-frequency cepstral coefficients (MFCCs) are specifically used as the main features because of how well they capture the spectral characteristics of speech. The most appropriate model for this task is determined by evaluating a number of classifiers, such as random forest classifiers (RF), convolutional neural networks (CNN), and support vector machines (SVM).[2]

This report describes our approach, the trials we ran, and the outcomes we got. We first review the relevant literature in the accent recognition field before going into great detail about our system design. After that, we describe our experimental design and findings, showing how well the system can identify regional accents from Palestine.

II. Background and related works

Research on accent recognition has been ongoing in the fields of computational linguistics and speech processing. Diverse approaches and strategies have been investigated to tackle the difficulties related to differentiating between accents. This section examines important related works that influenced the creation of our system for recognizing regional accents in Palestine.

1. Arabic Dialect Identification Using i-Vectors and ASR Transcripts

In-depth research on Arabic dialect identification was done by Ali et al. (2016), who addressed the complexity of Arabic dialects and how they differ greatly between geographical areas. The study concentrated on identifying various Arabic dialects using i-vectors and automatic speech recognition (ASR) transcripts; it offered insightful information and techniques that could be modified to identify regional accents in Palestinian Arabic.

In this study, the i-vector (identity vector) approach—which was first created for speaker verification—was successfully adapted for dialect identification. Fixed-length feature vectors called i-Vectors condense the most crucial details from variable-length speech segments. A universal background model (UBM) and a total variability model are trained in order to generate them. General speech characteristics are captured by the UBM, and the i-vectors are produced by mapping speech features to a low-dimensional space using the total variability model.

Ali et al. used a variety of machine learning models, such as neural networks and support vector machines (SVMs), for the classification task. These models were trained using the combined feature set, which included both i-vectors and ASR transcripts.

2. Accent Recognition in South African English

An important study on accent recognition in South African English was carried out by Kamper et al. (2015). This study examines the distinctive linguistic environment of South Africa, where a wide range of accents are used to speak English due to the nation's varied cultural and linguistic backgrounds. The study's approaches and conclusions, which examine useful techniques for differentiating between closely related accents, are especially pertinent to our work on Palestinian regional accent recognition.

In order to capture the distinctive qualities of various accents, Kamper et al. concentrated on extracting acoustic features from speech recordings. Mel-frequency cepstral coefficients (MFCCs), which are well known for their efficacy in speech processing tasks, were the main features employed. MFCCs were selected because of their capacity to depict a sound's short-term power spectrum, which is essential for encapsulating the subtleties of various accents.

The study used machine learning models to classify accents, including Gaussian Mixture Models (GMMs), Support Vector Machines (SVMs), and simple feedforward neural networks. These models effectively captured variability within each accent class, allowing for efficient classification. The performance of the system was evaluated using standard classification metrics like accuracy, precision, recall, and F1 score.

The efficiency of MFCCs as a feature representation for accent recognition was validated by the study. MFCCs achieved high classification accuracy by capturing the key acoustic characteristics that set one accent apart from another.

When it came to performance, the SVM and GMM models outperformed the neural network among the tested classifiers. Particularly, the SVM obtained the maximum accuracy, demonstrating its appropriateness for this kind of classification task.[5]

III. Methodology

In order to create the Palestinian regional accent recognition system, we implemented a thorough approach to feature extraction, data augmentation, and model selection. Through the application of sophisticated machine learning methods and extensive testing, we were able to develop a reliable system that could discriminate between the regional accents of Ramallah, Hebron, Jerusalem, and Nablus. Adding data augmentation and carefully adjusting hyperparameters were important components that improved our models' performance.

1. Dataset

There were two data sets one for training and one for testing, each one of them contain speech recordings from speakers representing the four primary Palestinian regional accents: Jerusalem, Nablus, Hebron, and Ramallah.

2. Feature Extraction

2.1 Acoustic Features

Mel-frequency cepstral coefficients are the main features used in accent recognition (MFCCs). Because MFCCs can capture the key components of speech signals, they are frequently used in speech processing. We extracted many MFCCs, which give a comprehensive picture of the audio's spectral characteristics, for every speech segment.

2.2 Data Augmentation

Several data augmentation techniques were used to increase the system system's reliability:

Adding Noise: To replicate actual circumstances, random noise was applied to the audio signals.

Time stretching: To produce variances in the speech tempo, the audio's speed was somewhat changed.

Pitch Shifting: To add variations in frequency, the audio's pitch was altered.

By adding diversity to the dataset, these augmentation techniques help the models perform better when applied to previously unseen data.

3. Model Selection

3.1 Random Forest Classifier

The classifier Random Forest (RF) was first used because of how well it handled high-dimensional feature spaces and how resilient it was to overfitting. An ensemble of decision trees, each trained on a different subset of the data, makes up the RF model. The total of all the trees' predictions is used to create the final forecast.

3.2 Hyperparameter Tuning

GridSearchCV hyperparameter tuning was used to maximize the RF classifier's performance. The number of trees (n_estimators) and the maximum depth of each tree (max_depth) were among the parameters that were adjusted. GridSearchCV uses cross-validation to assess each combination's performance while conducting a thorough search over the parameters that have been set.

3.3 Convolutional Neural Networks (CNNs)

As a result of CNNs' effectiveness in identifying regional patterns in speech signals, we also included one for accent recognition. The CNN architecture is made up of multiple convolutional layers that are followed by pooling layers. The purpose of the pooling layers is to use the input MFCC features to learn hierarchical feature representations.

3.4 Support Vector Machines (SVMs)

Using SVMs' ability to identify the ideal hyperplane dividing various accent classes, the application of SVMs for classification has been investigated. The radial basis function (RBF) kernel, which works well in high-dimensional spaces, was used to train the SVM model.

4. Model Training and Evaluation

4.1 Training Process

In order to minimize classification error, the model parameters were adjusted for each model during the training

process, which involved feeding the extracted MFCC features into the classifier. This involved standard training procedures for the RF and SVM models, and gradient descent and backpropagation for the CNN in order to optimize the network weights.

4.2 Validation and Testing

The validation set was used to verify the model's performance, and hyperparameter tuning was done to determine the best model configurations. After that, the testing set was used to assess the accuracy, precision, recall, and F1 score of the top-performing models.

4.3 Evaluation Metrics

Accuracy: The percentage of cases out of all instances that are correctly classified.

Precision can be defined as the percentage of all positive predictions that are true positive predictions.

Remember: The percentage of actual positives divided by the number of true positive predictions.

F1 Score: A single indicator of model performance that is derived from the harmonic mean of precision and recall.

IV. Experiments and results

1. Experiment one: support vector machine (SVM)

In this experiment, constants were set for sample rate (22050 Hz) which was chosen after experimenting two values and where the first was 22200 Hz, both were compared according to the accuracy produced, where the first assumption achieved was 38% accuracy, the number of MFCCs (Mel-frequency cepstral coefficients) (40) to capture an understandable representation of the speech signal's spectral properties, and the target cities with their corresponding labels. Then the `extract_features` function processed each audio file by loading it and applying data augmentation techniques, which includes adding noise, time stretching, and pitch shifting. MFCC features are then extracted from both the original and augmented audio and combined. If the combined MFCC length is shorter than a predefined maximum length, it is padded; otherwise, it is truncated, and the mean is taken across the time axis. Next, extracted the loaded data's MFCC features, and labeled them based on the filename. The data was then split into training and validation sets other than testing set which was originally prepared, and features are normalized using `StandardScaler`. The main SVM model was defined, and hyperparameter tuning was performed using `GridSearchCV` on the following hyperparameters: C (regularization parameter) and gamma (kernel coefficient). The best model was selected based on validation accuracy. Finally, the model's performance is evaluated on the test set, and results are printed, including the accuracy score and classification report. The results of this experiment are shown below.

```
Test Accuracy: 0.4
C:\Users\Administrator\AppData\Local\Programs\Python\Python312\Lib\
 to 0.0 in labels with no predicted samples. Use `zero_division` pa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(res
C:\Users\Administrator\AppData\Local\Programs\Python\Python312\Lib\
 to 0.0 in labels with no predicted samples. Use `zero_division` pa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(res
C:\Users\Administrator\AppData\Local\Programs\Python\Python312\Lib\
 to 0.0 in labels with no predicted samples. Use `zero_division` pa
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(res
              precision    recall  f1-score   support

ramallah-reef       0.29      1.00      0.45         5
    jerusalem       1.00      0.20      0.33         5
       hebron       0.00      0.00      0.00         5
       nablus       1.00      0.40      0.57         5

     accuracy                           0.40        20
    macro avg       0.57      0.40      0.34        20
 weighted avg       0.57      0.40      0.34        20
```

Figure IV- 1 evaluation results for SVM model

The SVM model achieved 40% test accuracy, which shows that the performance is moderate. The classification report shows that the model's performance varies significantly across different accents. For Ramallah reef, the recall is high at 1, but precision is low at 0.29, which indicates many false positives. For Jerusalem and Nablus, precision is perfect at 1, but recall is very low 0.20, which indicates many false negatives. For Hebron, both precision and recall are zero, which shows complete misclassification. The macro average F1-score is 0.34, showing overall poor balance between precision and recall across all classes. These results suggest the model struggles to generalize well across different accents.

2. Experiment two: Random Forest Classifier

The first step in this experiment is defining constants and a label mapping for four Palestinian cities. The same first steps in the process of SVM procedure, `extract_features` function loaded audio files, then applied noise addition, time stretching, and pitch shifting as data augmentation, and extracted MFCC features from these variations, then combined and padded them as needed. The dataset is split into training and validation sets, and features are normalized using `StandardScaler`, mentioning that the testing dataset is already prepared beforehand. Next, Random Forest was defined, and hyperparameter tuning was performed using `GridSearchCV` to find the best parameters. The parameters that were tuned are as follows: n_estimatros (number of random forests), and max_depth (maximum depth of each forest). The result of evaluation is shown below.

```
Test Accuracy: 0.75
              precision    recall  f1-score   support

ramallah-reef       0.56      1.00      0.71         5
    jerusalem       1.00      0.40      0.57         5
       hebron       0.83      1.00      0.91         5
       nablus       1.00      0.60      0.75         5

     accuracy                           0.75        20
    macro avg       0.85      0.75      0.74        20
 weighted avg       0.85      0.75      0.74        20
```

Figure IV- 2 evaluation results for Random Forest classifier

The Random Forest classifier achieved 75% test accuracy, which shows strong overall performance. The classification report shows that the model performed well on most classes but varies in precision and recall. Ramallah-reef has perfect recall of 1, but lower precision of 0.56, which indicates some false positives. Jerusalem has perfect precision at 1, but lower recall of 0.4, which indicates many false negatives. Both Hebron and Nablus show high precision (0.83 and 1, respectively) and good recall (1 and 0.6, respectively), with Hebron achieved the highest F1-score of 0.91. The macro and weighted averages of precision, recall, and F1-score (around 0.85, 0.75, and 0.74, respectively) highlights that the model generally performs well across all classes, though Jerusalem remains a more difficult class for accurate prediction, which is understandable since their accent has many words from all other classes.

3. Experiment three: convolutional neural network (CNN)

As with the previous tasks, audio features are extracted using the `librosa` library with data augmentation techniques such as adding noise, time stretching, and pitch shifting. MFCC features are computed and combined. The features are normalized by calculating the mean and standard deviation of the training data and applying the same normalization to the test data. Next, CNN model was defined with several layers including Conv1D, BatchNormalization, MaxPooling1D, Flatten, Dense, and Dropout layers. The model is compiled using the Adam optimizer. The following hyperparameters were tuned by trying different values: learning rate, epochs, batch. The result of the evaluation is shown below.

Accuracy: 0.5

	precision	recall	f1-score	support
ramallah-reef	1.00	0.40	0.57	5
jerusalem	0.67	0.40	0.50	5
hebron	0.60	0.60	0.60	5
nablus	0.30	0.60	0.40	5
accuracy			0.50	20
macro avg	0.64	0.50	0.52	20
weighted avg	0.64	0.50	0.52	20

Figure IV- 3 evaluation results for CNN

The results indicate that the CNN achieved an overall accuracy of 0.5. In the classification of different categories, the precision for Ramallah reef was 1, which indicates that when the model predicted this category, it was correct all the time, while the recall was 0.4, which suggests that it identified only 40% of the instances of their accent. For Jerusalem, precision was 0.67, recall was 0.4, and F1-score was 0.5. Hebron had precision of 0.6, recall of 0.6, and F1-score of 0.6, which shows relatively balanced performance. However, for Nablus, precision was 0.3, recall was 0.6, and F1-score was 0.4, which shows lower precision and F1-score compared to recall. The macro and weighted averages of precision, recall, and F1-score are also provided, with macro average precision, recall, and F1-score being 0.64, 0.5, and 0.52 respectively, while weighted averages were the same as macro averages, which indicates uniformity in class distribution. Overall, the results show that the CNN's performance is moderate but not as less as acceptable.

V. CONCLUSION AND FUTURE WORK

In conclusion, we developed and evaluated an accent recognition system for Palestinian regional accents using both traditional machine learning models (SVM and Random Forest) and Convolutional Neural Network (CNN). Our main findings showed that the Random Forest classifier achieved the highest test accuracy of 75%, which means strong overall performance in recognizing accents from Ramallah-reef, Jerusalem, Hebron, and Nablus. The classification results highlighted varying degrees of precision and recall across different accents, with Hebron achieving the highest F1-score. The CNN model, trained with augmented data and sophisticated feature extraction techniques, also showed moderate and promising results (50%) accuracy, which leads us to think about the potential of deep learning approaches in this domain (if we have more dataset entries with longer time and have more phrases and words that may be regarded as the main key between each accent). The poorest model was SVM that achieved (40%) accuracy and poor results in other evaluation metrics even after hyperparameter tuning, so there will be no more development on this model.

For future work, many approaches can be employed to enhance the system performance. First, increasing the size and diversity of the training dataset may help improve the model's ability to generalize across different speakers and recording conditions. Second, experimenting with more advanced data augmentation techniques and feature extraction methods, such as spectrograms and using transfer learning from pre-trained models, could achieve better representations of the speech signal.

VI. PARTNER PARTICIPATION TASKS

Our project was a result of our collaborative effort between us, Amal and Bara. Amal took the lead on developing the SVM model with its hyperparameter tuning. Bara was responsible for the development and optimization of the Random Forest model. Both of us worked together on the implementation and training of the CNN, pooling our expertise to enhance the model's performance. For the report, Bara handled the introduction, methodology, and related works sections, providing beneficial foundation for the project. Amal focused on writing the abstract, explaining the procedure of the experiments, and writing the conclusion which summarizes the main findings and suggests future improvements.

VII. REFERENCES

[1]: Behnke, M., He, Y., Tomalin, M., & Stevenson, M. (2017). *Automatic Dialect and Accent Recognition and its Application to Speech Recognition*. Computer Speech & Language, 46, 1-17. doi:10.1016/j.csl.2017.04.004.

[2]: Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., ... & Kingsbury, B. (2012). *Deep Learning for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups*. IEEE Signal Processing Magazine, 29(6), 82-97. doi:10.1109/MSP.2012.2205597.

[3]: Ali, A., Menon, R., Yimam, S. M., & Vogel, S. (2016). *Arabic Dialect Identification Using iVectors and ASR Transcripts*. Proceedings of Interspeech 2016, 2934-2938. doi:10.21437/Interspeech.2016-1166.

[5]: Kamper, H., Kleynhans, N., & Niesler, T. (2015). *Accent Recognition for South African English*. Proceedings of Interspeech 2015, 2157-2161. doi:10.21437/Interspeech.2015-486.

VIII. APPENDIX

1. SVM code implementation

```python
import os
import numpy as np
import librosa
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split, GridSearchCV

# Constants
DATA_DIR = r'C:\Users\Administrator\spoken project'
NUM_MFCC = 40  # Number of MFCCs
SAMPLE_RATE = 22050
MAX_LEN = 216  # Maximum length of the MFCC features (determined empirically)
CITIES = ['ramallah-reef', 'jerusalem', 'hebron', 'nablus']
CITY_TO_LABEL = {city: idx for idx, city in enumerate(CITIES)}

def extract_features(file_path):
    y, sr = librosa.load(file_path, sr=SAMPLE_RATE)

    # Data augmentation: adding noise
    noise = np.random.randn(len(y))
    y_noisy = y + 0.005 * noise

    # Data augmentation: time stretching
    y_stretched = librosa.effects.time_stretch(y, rate=0.9)

    # Data augmentation: pitch shifting
    y_shifted = librosa.effects.pitch_shift(y, sr=sr, n_steps=2)

    # MFCC features
    mfcc = librosa.feature.mfcc(y=y_noisy, sr=sr, n_mfcc=NUM_MFCC)
    mfcc_stretched = librosa.feature.mfcc(y=y_stretched, sr=sr, n_mfcc=NUM_MFCC)
    mfcc_shifted = librosa.feature.mfcc(y=y_shifted, sr=sr, n_mfcc=NUM_MFCC)

    # Combine the original and augmented MFCCs
    mfcc_combined = np.concatenate([mfcc, mfcc_stretched, mfcc_shifted], axis=1)
```

```python
def extract_features(file_path):

    if mfcc_combined.shape[1] < MAX_LEN:
        pad_width = MAX_LEN - mfcc_combined.shape[1]
        mfcc_combined = np.pad(mfcc_combined, pad_width=((0, 0), (0, pad_width)), mode='constant')
    else:
        mfcc_combined = mfcc_combined[:, :MAX_LEN]

    mfcc_combined = np.mean(mfcc_combined.T, axis=0)
    return mfcc_combined

def load_data(data_dir):
    features = []
    labels = []
    for file_name in os.listdir(data_dir):
        if file_name.endswith('.wav'):
            file_path = os.path.join(data_dir, file_name)
            mfcc = extract_features(file_path)
            features.append(mfcc)
            label_str = file_name.split('_')[0]
            labels.append(CITY_TO_LABEL[label_str])
    return np.array(features), np.array(labels)

# Load data
X, y = load_data(os.path.join(DATA_DIR, 'training'))
X_test, y_test = load_data(os.path.join(DATA_DIR, 'testing'))

# Normalize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_test = scaler.transform(X_test)

# Split training data into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Define the models
svm_model = SVC()

# Hyperparameter tuning for SVM
svm_param_grid = {'C': [0.1, 1, 10, 100], 'gamma': [1, 0.1, 0.01, 0.001]}
svm_grid = GridSearchCV(svm_model, svm_param_grid, refit=True, verbose=2, cv=5)
svm_grid.fit(X_train, y_train)

# Evaluate the models
svm_best = svm_grid.best_estimator_

# Predict on the validation set
svm_val_pred = svm_best.predict(X_val)

print("SVM Validation Accuracy:", accuracy_score(y_val, svm_val_pred))

best_model = svm_best

y_test_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_test_pred)

print(f'Test Accuracy: {accuracy}')
print(classification_report(y_test, y_test_pred, target_names=CITIES))
```

2. Random forest code implementation

```python
import os
import numpy as np
import librosa
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split, GridSearchCV

# Constants
DATA_DIR = r'C:\Users\Administrator\spoken project'
NUM_MFCC = 40  # Number of MFCCs
SAMPLE_RATE = 22050
MAX_LEN = 216  # Maximum length of the MFCC features (determined empirically)
CITIES = ['ramallah-reef', 'jerusalem', 'hebron', 'nablus']
CITY_TO_LABEL = {city: idx for idx, city in enumerate(CITIES)}

def extract_features(file_path):
    y, sr = librosa.load(file_path, sr=SAMPLE_RATE)

    # Data augmentation: adding noise
    noise = np.random.randn(len(y))
    y_noisy = y + 0.005 * noise

    # Data augmentation: time stretching
    y_stretched = librosa.effects.time_stretch(y, rate=0.9)

    # Data augmentation: pitch shifting
    y_shifted = librosa.effects.pitch_shift(y, sr=sr, n_steps=2)

    # MFCC features
    mfcc = librosa.feature.mfcc(y=y_noisy, sr=sr, n_mfcc=NUM_MFCC)
    mfcc_stretched = librosa.feature.mfcc(y=y_stretched, sr=sr, n_mfcc=NUM_MFCC)
    mfcc_shifted = librosa.feature.mfcc(y=y_shifted, sr=sr, n_mfcc=NUM_MFCC)

    # Combine the original and augmented MFCCs
    mfcc_combined = np.concatenate([mfcc, mfcc_stretched, mfcc_shifted], axis=1)
```

```python
def extract_features(file_path):

    if mfcc_combined.shape[1] < MAX_LEN:
        pad_width = MAX_LEN - mfcc_combined.shape[1]
        mfcc_combined = np.pad(mfcc_combined, pad_width=((0, 0), (0, pad_width)), mode='constant')
    else:
        mfcc_combined = mfcc_combined[:, :MAX_LEN]

    mfcc_combined = np.mean(mfcc_combined.T, axis=0)
    return mfcc_combined

def load_data(data_dir):
    features = []
    labels = []
    for file_name in os.listdir(data_dir):
        if file_name.endswith('.wav'):
            file_path = os.path.join(data_dir, file_name)
            mfcc = extract_features(file_path)
            features.append(mfcc)
            label_str = file_name.split('_')[0]
            labels.append(CITY_TO_LABEL[label_str])
    return np.array(features), np.array(labels)

# Load data
X, y = load_data(os.path.join(DATA_DIR, 'training'))
X_test, y_test = load_data(os.path.join(DATA_DIR, 'testing'))

# Normalize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_test = scaler.transform(X_test)

# Split training data into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

```python
# Define the models
rf_model = RandomForestClassifier()


# Hyperparameter tuning for Random Forest
rf_param_grid = {'n_estimators': [50, 100, 200], 'max_depth': [10, 20, None]}
rf_grid = GridSearchCV(rf_model, rf_param_grid, refit=True, verbose=2, cv=5)
rf_grid.fit(X_train, y_train)

# Evaluate the models
rf_best = rf_grid.best_estimator_

# Predict on the validation set
rf_val_pred = rf_best.predict(X_val)

y_test_pred = rf_best.predict(X_test)
accuracy = accuracy_score(y_test, y_test_pred)

print(f'Test Accuracy: {accuracy}')
print(classification_report(y_test, y_test_pred, target_names=CITIES))
```

3. CNN code implementation

```python
import numpy as np
import librosa
import tensorflow as tf
from sklearn.metrics import classification_report, accuracy_score
from keras import layers
import os

# Constants
DATA_DIR = r'C:\Users\Administrator\spoken project'
NUM_MFCC = 40  # Number of MFCCs
SAMPLE_RATE = 22050
MAX_LEN = 216  # Maximum length of the MFCC features (determined empirically)
CITIES = ['ramallah-reef', 'jerusalem', 'hebron', 'nablus']
CITY_TO_LABEL = {city: idx for idx, city in enumerate(CITIES)}

def extract_features(file_path):
    y, sr = librosa.load(file_path, sr=SAMPLE_RATE)

    # Data augmentation: adding noise
    noise = np.random.randn(len(y))
    y_noisy = y + 0.005 * noise

    # Data augmentation: time stretching
    y_stretched = librosa.effects.time_stretch(y, rate=0.9)

    # Data augmentation: pitch shifting
    y_shifted = librosa.effects.pitch_shift(y, sr=sr, n_steps=2)

    # MFCC features
    mfcc = librosa.feature.mfcc(y=y_noisy, sr=sr, n_mfcc=NUM_MFCC)
    mfcc_stretched = librosa.feature.mfcc(y=y_stretched, sr=sr, n_mfcc=NUM_MFCC)
    mfcc_shifted = librosa.feature.mfcc(y=y_shifted, sr=sr, n_mfcc=NUM_MFCC)

    # Combine the original and augmented MFCCs
    mfcc_combined = np.concatenate([mfcc, mfcc_stretched, mfcc_shifted], axis=1)
```

```
def extract_features(file_path):
    if mfcc_combined.shape[1] < MAX_LEN:
        pad_width = MAX_LEN - mfcc_combined.shape[1]
        mfcc_combined = np.pad(mfcc_combined, pad_width=((0, 0), (0, pad_width)), mode='constant')
    else:
        mfcc_combined = mfcc_combined[:, :MAX_LEN]

    mfcc_combined = np.mean(mfcc_combined.T, axis=0)
    return mfcc_combined

def load_data(data_dir):
    features = []
    labels = []
    for file_name in os.listdir(data_dir):
        if file_name.endswith('.wav'):
            file_path = os.path.join(data_dir, file_name)
            mfcc = extract_features(file_path)
            features.append(mfcc)
            label_str = file_name.split('_')[0]
            labels.append(CITY_TO_LABEL[label_str])
    return np.array(features), np.array(labels)

# Load training data
X_train, y_train = load_data(os.path.join(DATA_DIR, 'training'))

# Load testing data
X_test, y_test = load_data(os.path.join(DATA_DIR, 'testing'))

# Normalize the features
mean = np.mean(X_train, axis=0)
std = np.std(X_train, axis=0)
X_train = (X_train - mean) / std
X_test = (X_test - mean) / std
```

```
# Define the model using CNN
model = tf.keras.Sequential([
    tf.keras.layers.Reshape((NUM_MFCC, 1), input_shape=(NUM_MFCC,)),
    tf.keras.layers.Conv1D(64, kernel_size=3, activation='relu'),
    layers.BatchNormalization(),
    tf.keras.layers.MaxPooling1D(pool_size=2),
    tf.keras.layers.Conv1D(128, kernel_size=3, activation='relu'),
    layers.BatchNormalization(),
    tf.keras.layers.MaxPooling1D(pool_size=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(len(CITIES), activation='softmax')
])

# Compile the model with a lower learning rate
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model with a larger batch size and more epochs
model.fit(X_train, y_train, epochs=150, batch_size=8, validation_split=0.2)

# Evaluate the model
y_pred = np.argmax(model.predict(X_test), axis=1)
accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')
print(classification_report(y_test, y_pred, target_names=CITIES))
```