Acoustic Palestinian Regional Accent Recognition System

Electrical and Computer Engineering Department
SPOKEN LANGUAGE PROCESSING (ENCS5344)

Musab Masalmah, 1200078
Jehad Hamayel, 1200348
Abdalkarim Eiss, 1200015

Abualsoud Hanani
Electrical and Computer Engineering
ahanani@birzeit.edu

Abstract

In this study, a basic system for recognizing the regional accent of Palestinians through audio is developed and evaluated. There is a great deal of variation in Palestinian accents, with different sub-accents for places like Ramallah, Jerusalem, Nablus, and Hebron. This project's main objective is to create a system that can use brief speech segments to identify a speaker's accent from any of these four regions. The system does accent recognition by using acoustic cues that are derived from the speech. The first step in the procedure is gathering voice data that represents each of the four regional accents. Preprocessing the audio data in order to reduce noise and improve recording quality is the next stage, to extract the unique qualities of each accent, feature extraction techniques like spectral features and Mel-Frequency Cepstral Coefficients (MFCCs) are used. The effectiveness of many machine learning techniques in categorizing the accents is then assessed through training and evaluation, including Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Neural Networks. The models' performance is evaluated through the use of common measures including F1-score, accuracy, precision, and recall. The outcomes show that automatic accent recognition is feasible. By offering insights into the creation of reliable accent identification algorithms that may be expanded to additional dialects and languages, this effort advances the field of voice processing.

Index Terms: Palestinian accents, speech processing, feature extraction, machine learning.

1. Introduction

This paper explores the process of creating a system for identifying regional accents in Palestinian. The accents of Palestinian cities such as Ramallah, Hebron, Nablus, and Jerusalem are diverse. Among these variances, the study seeks to precisely detect the speaker's accent. We discuss the significance of accent recognition and our methodology for developing systems. The difficulties and advancements in the identification of Palestinian accents are addressed in this paper.

We use an extensive methodology that combines state-of-the-art methods for feature extraction, machine learning, and voice processing to construct our accent identification system. To capture the unique qualities of each accent, acoustic parameters like Mel-Frequency Cepstral Coefficients (MFCCs) are retrieved from the speech samples. The accents are then reliably classified by using these attributes to train a variety of machine learning models, such as Support Vector Machines (SVMs) and Neural Networks [1].

2. Background / Related Work:

Developing a workable solution requires an understanding of the state of accent recognition software and the subtleties of regional accents in Palestine. A review of earlier studies in the topic and the unique difficulties presented by Palestinian accents are given in this section.

2.1. Palestinian Regional Accents

The Palestinian accent is considered one of the broad classes of Levantine Arabic dialects, which includes Arabic dialects of people in Syria, Lebanon, Jordan and Palestine. However, the Palestinian accent contains huge variations, and it can be divided into subsets of accents. We decided to divide Palestinian accents regionally. Each region in Palestine can be assumed to have its own sub accent. Since covering all Arab regions in Palestine is difficult, we decided to consider only four major regions in this study: Jerusalem, Hebron, Nablus, and rural areas around Ramallah. Accents in these four regions are distinctively distinguishable and well known in Palestine [2].

2.2. Accent Recognition Systems

Acoustic modeling, feature extraction, and machine learning are some of the approaches used by accent recognition systems to recognize and categorize accents. Using deep learning and adaptive approaches, these systems have made headway toward their aim of enhancing voice recognition accuracy. Convolutional Neural Networks (CNNs) have proven particularly useful at capturing the intricacies of various accents. Furthermore, incorporating accent information into automated speech recognition (ASR) systems helps create more robust models that can handle a wide range of speech patterns [3].

2.3. Challenges in Accent Recognition

Accent Recognition has a number of obstacles, including pronunciation variability, limited access to annotated voice data, and the necessity for adequate feature representation to capture accent-specific traits. Differences in pronunciation within and between accent groups confound classification. Furthermore, there is sometimes a scarcity of high-quality, labeled datasets for training and evaluating accent detection algorithms, impeding the development of reliable systems. Effective feature extraction methods are critical for effectively distinguishing between small accent differences and improving recognition performance. Addressing these problems is critical for developing the discipline and improving the accuracy and reliability of accent recognition systems [3].

3. Methodology

This section describes the methods for developing the Palestinian regional accent identification system, which includes feature extraction, normalization of data, and machine learning algorithms. The method comprises extracting significant acoustic properties from speech samples, normalizing these features, and using them to train machine learning models for accent classification.

3.1. Feature Extraction

Feature extraction is an important stage in the development of voice recognition systems. For this research, we used the librosa package to extract several acoustic properties from audio recordings. Mel-Frequency Cepstral Coefficients (MFCCs), energy, and their delta and delta-delta derivatives are the most commonly employed characteristics.

3.1.1. Mel-Frequency Cepstral Coefficients (MFCCs)

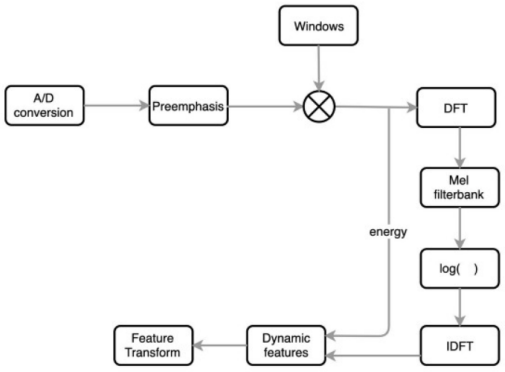

Figure 1: MFCC System [4]

A/D Conversion: In this step, we will convert our audio signal from analog to digital format with a sampling frequency of 8kHz or 16kHz [4].

Pre-emphasis: enhances the magnitude of energy in higher frequencies. When we examine the frequency domain of an audio signal for voiced segments such as vowels, we see that the energy at higher frequencies is significantly smaller than the energy at lower frequencies. Increasing the energy at higher frequencies improves phone detection accuracy, which improves the model's performance [4].

Windowing: To ensure stable acoustic analysis, speech signals are divided into frames lasting 20-30ms, aligning with quasi-stationary segments. Vowel sounds, however, span 40ms to 80ms. Frames are typically 20ms long with 10ms overlaps, facilitating tracking of temporal speech characteristics. Overlapping frames centers speech representation, enhancing analysis accuracy [5].

DFT (Discrete Fourier Transform): It will convert the signal from the time domain to the frequency domain by applying the dft transform. For audio signals, analyzing in the frequency domain is easier than in the time domain [4].

The Mel Filter Bank: is a bank of filters, which is constructed based on pitch perception. The Mel filter was originally developed for speech analysis and like human ear perceiving of speech, it targets extracting non-linear representation of the speech signal [5].

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \le k < f(m) \\ 1 & k = f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) < k \le f(m+1) \\ 0 & k > f(m+1) \end{cases}$$

Figure 2: Mel Filter Equation [5]

Applying Log: Humans are less sensitive to changes in audio signal energy at higher levels compared to lower levels. The log function has a similar trait in that the gradient of the log function is larger when the input value is low, but lower when the input value is high. So we apply log to the Mel-filter output to simulate the human hearing system [4].

IDFT: In this step, we are doing the inverse transform of the output from the previous step. Before knowing why we have to do inverse transform we have to first understand how the sound produced by human beings [4].

Dynamic Features: MFCC technique will generate 39 features from each audio signal sample which are used as input for the speech recognition model [4].

3.1.2. Energy

The energy feature represents the amplitude of the audio signal and indicates the volume of the speech. It is calculated using the magnitude of the STFT [2].

3.1.3. Delta and Delta-Delta Features

Delta features depict the rate of change in MFCCs and energy across time, whereas delta-delta features capture the acceleration of these changes. These qualities are essential for understanding the dynamic dynamics of speech because they reveal how the spectral characteristics evolve [3].

3.2. Data Normalization

To confirm that the collected characteristics were appropriate for training machine learning models, we normalized the data. Normalization aids in scaling the characteristics to a standard range, hence improving the performance of many machine learning methods. In this process, we used mean normalization, which adjusts the features by subtracting the mean and dividing by the range, ensuring that they are centered around zero [2].

3.3. Machine Learning

During the machine learning phase, three classifiers were used: K-Nearest Neighbors (KNN), Support Vector Machines (SVM), and Random Forest Classifiers (RFC). These classifiers were trained and tested with normalized features. KNN is a basic but effective nonparametric classifier that

gives a class to a sample based on the majority vote of its k nearest neighbors in the feature space. We utilized KNN with k=4 to get a fair balance of bias and variance [6].

SVMs are effective classifiers that determine the best hyper plane to divide various classes, especially in high-dimensional domains. For this project, we employed grid search with cross-validation to modify hyper parameters such as kernel type (linear, polynomial, radial basis function, sigmoid) and polynomial kernel degree, improving the SVM for accent recognition. RFC is an ensemble classifier that builds numerous decision trees during training and outputs the mode class for each tree. The approach prevents over fitting by averaging numerous trees, and we used grid search and cross-validation to find the optimal hyper parameters, such as the number of trees and maximum depth [6].

4. Experiments and Results

This section describes the experiments that done to evaluate the performance of our Palestinian regional accent recognition system. We start by detailing the dataset into training and testing datasets, followed by feature extraction process. After that, we describe the machine learning models which are applied and the evaluation metrics employed to measure the system's accuracy. Finally, we provide an analysis of the whole system's results including both quantitative and qualitative evaluations, to demonstrate the effectiveness of our approach.

4.1. Dataset

The datasets composed of voice files which were collected into two main files: training data and testing data. Each one includes number of secondary files (ex: Ramallah_Reef, Nablus and etc), each of them includes multiple voice files in .wav extension. The features that were extracted from these voice files into CSV files which were collected later to form a general data. These data were used for training and evaluating.

4.2. Evaluation Scales

The evaluation scale was based on the standard classification metrics: recall, precision, F1-score, and accuracy. Additionally, grid search with cross-validation was employed to optimize hyperparameters for each model, ensuring robust performance evaluation.

4.3. Training and Testing

In these experiments, three classifiers were used. These classifiers include K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Random Forest Classifier (RFC). Each classifier was trained by the features that extracted from training dataset. Also, each classifier was tested by the features that extracted from testing dataset. The KNN classifier was tested with multiple k values while SVM and RFC underwent hyperparameter tuning through grid search.

4.4. Results

The results describe the performance of the three different classifiers: KNN, SVM and RFC. For KNN, when k=4 was the best results, also the resulted accuracy was 25%. Table 1 shows the detailed performance metrics.

Class	Precision	Recall	F1-Score	Support
Hebron	25%	40%	31%	5
Jerusalem	17%	20%	18%	5
Nablus	100%	20%	33%	5
Ramallah_Reef	20%	20%	20%	5
Accuracy			25%	20
Macro avg	40%	25%	26%	20
Weighted avg	40%	25%	26%	20

Table 1: KNN Results

Also, for SVM, which was optimized using grid search with cross-validation, identified the best parameters as {degree: 2, kernel: 'linear'} and achieved an accuracy of 70% with a best cross-validation score of 60%. Table 2 shows the detailed performance metrics.

Class	Precision	Recall	F1-Score	Support
Hebron	80%	80%	80%	5
Jerusalem	83%	100%	91%	5
Nablus	67%	40%	50%	5
Ramallah_Reef	50%	60%	55%	5
Accuracy			70%	20
Macro avg	70%	70%	69%	20
Weighted avg	70%	70%	69%	20

Table 2: SVM Results

Also, for RFC, which was optimized using grid search, identified the best parameters as {max depth: 5, n_estimators: 200} and achieved an accuracy of 35% with a best cross-validation score of 65%. Table 3 shows the detailed performance metrics.

Class	Precision	Recall	F1-Score	Support
Hebron	30%	60%	40%	5
Jerusalem	40%	40%	40%	5
Nablus	50%	20%	29%	5
Ramallah_Reef	0%	0%	0%	5
Accuracy			30%	20
Macro avg	30%	30%	27%	20
Weighted avg	30%	30%	27%	20

Table 3: RFC Results

Comprehensive confusion matrices for these classifiers provided information about how well they performed in various models. For example, the SVM performed well across models, with many right predictions. However, the RFC and KNN performed in contradiction well. Figure 3, Figure 4 and Figure 5 depict these quantitative assessments, presenting a visual comparison of actual versus expected labels alongside the corresponding confusion matrices for KNN, SVM, and RFC, respectively.

```
KNN Classifier with K=4 Accuracy: 0.25
                precision    recall  f1-score   support

        Hebron       0.25      0.40      0.31         5
     Jerusalem       0.17      0.20      0.18         5
        Nablus       1.00      0.20      0.33         5
Ramallah_Reef        0.20      0.20      0.20         5

      accuracy                           0.25        20
     macro avg       0.40      0.25      0.26        20
  weighted avg       0.40      0.25      0.26        20

Real: Ramallah_Reef, Predicted: Hebron
Real: Ramallah_Reef, Predicted: Hebron
Real: Ramallah_Reef, Predicted: Jerusalem
Real: Ramallah_Reef, Predicted: Ramallah_Reef
Real: Ramallah_Reef, Predicted: Hebron
Real: Nablus, Predicted: Nablus
Real: Nablus, Predicted: Jerusalem
Real: Nablus, Predicted: Hebron
Real: Nablus, Predicted: Jerusalem
Real: Nablus, Predicted: Ramallah_Reef
Real: Jerusalem, Predicted: Jerusalem
Real: Jerusalem, Predicted: Hebron
Real: Jerusalem, Predicted: Hebron
Real: Jerusalem, Predicted: Ramallah_Reef
Real: Jerusalem, Predicted: Ramallah_Reef
Real: Hebron, Predicted: Hebron
Real: Hebron, Predicted: Jerusalem
Real: Hebron, Predicted: Hebron
Real: Hebron, Predicted: Jerusalem
Real: Hebron, Predicted: Ramallah_Reef
```

Figure 3: KNN Confusion Matrices

```
Best Parameters for SVM: {'degree': 2, 'kernel': 'linear'}
Best Cross-Validation Score: 0.6
Accuracy of SVM : 0.7
                precision    recall  f1-score   support

        Hebron       0.80      0.80      0.80         5
     Jerusalem       0.83      1.00      0.91         5
        Nablus       0.67      0.40      0.50         5
Ramallah_Reef        0.50      0.60      0.55         5

      accuracy                           0.70        20
     macro avg       0.70      0.70      0.69        20
  weighted avg       0.70      0.70      0.69        20

Real: Ramallah_Reef, Predicted: Ramallah_Reef
Real: Ramallah_Reef, Predicted: Ramallah_Reef
Real: Ramallah_Reef, Predicted: Ramallah_Reef
Real: Ramallah_Reef, Predicted: Nablus
Real: Ramallah_Reef, Predicted: Hebron
Real: Nablus, Predicted: Nablus
Real: Nablus, Predicted: Ramallah_Reef
Real: Nablus, Predicted: Jerusalem
Real: Nablus, Predicted: Nablus
Real: Nablus, Predicted: Ramallah_Reef
Real: Jerusalem, Predicted: Jerusalem
Real: Jerusalem, Predicted: Jerusalem
Real: Jerusalem, Predicted: Jerusalem
Real: Jerusalem, Predicted: Jerusalem
Real: Jerusalem, Predicted: Jerusalem
Real: Hebron, Predicted: Hebron
Real: Hebron, Predicted: Ramallah_Reef
Real: Hebron, Predicted: Hebron
Real: Hebron, Predicted: Hebron
Real: Hebron, Predicted: Hebron
```

Figure 4: SVM Confusion Matrices

```
Best Parameters for RFC: {'max_depth': 5, 'n_estimators': 200}
Best Cross-Validation Score: 0.65
Accuracy of RFC : 0.3
                precision    recall  f1-score   support

        Hebron       0.30      0.60      0.40         5
     Jerusalem       0.40      0.40      0.40         5
        Nablus       0.50      0.20      0.29         5
Ramallah_Reef        0.00      0.00      0.00         5

      accuracy                           0.30        20
     macro avg       0.30      0.30      0.27        20
  weighted avg       0.30      0.30      0.27        20

Real: Ramallah_Reef, Predicted: Hebron
Real: Ramallah_Reef, Predicted: Hebron
Real: Ramallah_Reef, Predicted: Jerusalem
Real: Ramallah_Reef, Predicted: Nablus
Real: Ramallah_Reef, Predicted: Hebron
Real: Nablus, Predicted: Nablus
Real: Nablus, Predicted: Jerusalem
Real: Nablus, Predicted: Ramallah_Reef
Real: Nablus, Predicted: Hebron
Real: Nablus, Predicted: Hebron
Real: Jerusalem, Predicted: Hebron
Real: Jerusalem, Predicted: Jerusalem
Real: Jerusalem, Predicted: Hebron
Real: Jerusalem, Predicted: Jerusalem
Real: Jerusalem, Predicted: Ramallah_Reef
Real: Hebron, Predicted: Hebron
Real: Hebron, Predicted: Ramallah_Reef
Real: Hebron, Predicted: Hebron
Real: Hebron, Predicted: Jerusalem
Real: Hebron, Predicted: Hebron
```

Figure 5: RFC Confusion Matrices

5. Conclusion

In the conclusion, our system demonstrates the ability of recognizing Palestinian regional accents with high accuracy using acoustic features and machine learning models. For KNN, the best k-value when k-value of 4 yielded with an accuracy of 25%. Also, SVM was achieved an accuracy of 70% with a best cross-validation score of 60%. Also, RFC was achieved an accuracy of 35% with a best cross-validation score of 65%. These results highlight the importance of parameter tuning in improving classifier prediction performance, with SVM showing the highest accuracy with RFC and KNN following suit. But the datasets are little. To further enhance performance., future works could focus on incorporating more diverse datasets and exploring advanced neural network architectures.

6. Team work

We worked on the entire project collectively, and each of us had work on every part of the project, as we worked to hold Zoom meetings and work together on every part of the project in an equal and fair manner, so one of us was applying his work to his own device and repeating the thing for everyone in order to benefit. And understand the project more.

7. References

[1] "Wikipedia," 2024. [Online]. Available: https://en.wikipedia.org/wiki/Support_vector_machine. [Accessed 6 June 2024].

[2] Hanani, A., Basha, H., Sharaf, Y., & Taylor, S, "Palestinian Arabic Regional Accent Recognition, International Conference on Speech Technology and Human-Computer Dialogue (SpeD)," 2015.

[3] Y. &. L. X. Zhu, "Accented Speech Recognition," 2021. [Online]. Available: https://ar5iv.labs.arxiv.org/html/2104.10747. [Accessed 6 June 2024].

[4] "Analytics Vidhya, MFCC Technique for Speech Recognition," June 2021. [Online]. Available: https://www.analyticsvidhya.com/blog/2021/06/mfcc-technique-for-speech-recognition/. [Accessed 7 June 2024].

[5] "IEEE Xplore," [Online]. Available: https://ieeexplore.ieee.org/document/9955539. [Accessed 7 June 2024].

[6] "Research Gate," [Online]. Available: https://www.researchgate.net/figure/Comparative-study-different-Classification-algorithm-RFC-KNN-SVM-Deep-belief_tbl3_345768928. [Accessed 7 June 2024].

8. Appendix

This section contains all codes implementation that used in this system. All implementations were written in Python language and using Colab tool.

8.1. Feature Extraction Code

```
import librosa
```

```
import numpy as np
import os
import pandas as pd
from google.colab import drive


import librosa
import numpy as np

def feature_extraction(file_path):
    # Load your audio file
    y, sr = librosa.load(file_path)

    pre_emphasis = 0.97
    # Pre-emphasis
    y = np.append(y[0], y[1:] -
pre_emphasis * y[:-1])

    window_size = 0.025  # 25ms
window size
    hop_size = 0.010  # 10ms hop
size
    # Window and hop length in
samples
    n_fft = int(window_size * sr)  #
25ms window size
    hop_length = int(hop_size * sr)
 # 10ms hop size

    # Compute the Short-Time Fourier
Transform (STFT)
    Short_Time_Fourier_Transform =
librosa.stft(y, n_fft=n_fft,
hop_length=hop_length)
    S, _ =
librosa.magphase(Short_Time_Fourier_
Transform)

    # Compute the energy feature
from the magnitude spectrogram
    energy =
librosa.feature.rms(S=S,
frame_length=n_fft,
hop_length=hop_length)

    # Extract the Mel filter bank
features and compute log-mel
spectrogram
    mel_filter_spectrogram =
librosa.feature.melspectrogram(S=S,
sr=sr, n_fft=n_fft,
hop_length=hop_length)
    log_mel_spectrogram =
librosa.power_to_db(mel_filter_spect
rogram)

    # Extract 12 MFCCs (excluding
the 0th coefficient)
    mfccs =
librosa.feature.mfcc(S=log_mel_spect
rogram, sr=sr, n_mfcc=12)

    # Calculate the delta and delta-
delta features for MFCCs
    delta_mfccs =
librosa.feature.delta(mfccs,
order=1)
    double_delta_mfccs =
librosa.feature.delta(mfccs,
order=2)

    # Calculate the delta and delta-
delta for energy
    delta_energy =
librosa.feature.delta(energy,
order=1)
    double_delta_energy =
librosa.feature.delta(energy,
order=2)

    # Concatenate MFCCs and energy
along with their delta and delta-
delta features
    combined_features = np.vstack([
        mfccs,
        energy,
        delta_mfccs,
        delta_energy,
        double_delta_mfccs,
        double_delta_energy
    ])

    # Ensure you have 39 total
features: 13 (12 MFCC + 1 energy) *
3 = 39
```

```
    assert
combined_features.shape[0] == 39,
f"Number of features does not match
39, but
{combined_features.shape[0]}"

    return combined_features

root_dir = ['/content/drive/My
Drive/training data',
'/content/drive/My Drive/testing
data']
for root in  root_dir:
    # Iterate through all
directories and files
    for subdir, dirs, files in
os.walk(root):


        for file in files:
            if
file.endswith('.wav'):
                filepath =
os.path.join(subdir, file)
                print(f'Processing
{filepath}')
                features =
feature_extraction(filepath)


print(features.shape)


                # Save features to
CSV

                feature_df =
pd.DataFrame(features).T  #
Transpose to have frames as rows
                csv_filename =
os.path.splitext(filepath)[0] +
'_features.csv'

feature_df.to_csv(csv_filename,
index=False)
                print(f'Saved
features to {csv_filename}')
```

8.2. Data Aggregation and Target Extraction Code

```
import os
import pandas as pd
import numpy as np

# Directory containing folders
big_directorys = ['/content/drive/My
Drive/training data/',
'/content/drive/My Drive/testing
data/']

for big_directory in big_directorys:
    # Initialize an empty list to
store aggregated features and
targets
    all_data = []

    # Iterate through each folder in
the big directory
    for folder_name in
os.listdir(big_directory):
        folder_path =
os.path.join(big_directory,
folder_name)
        if
os.path.isdir(folder_path):
            # Iterate through each
file in the folder
            for filename in
os.listdir(folder_path):
                if
filename.endswith(".csv"):
                    # Load the data
                    file_path =
os.path.join(folder_path, filename)
                    df =
pd.read_csv(file_path)

                    # Aggregate
Features
                    mean_features =
df.mean(axis=0)

                    # Extract target
from folder name
                    target =
folder_name
```

```
                    # Append
aggregated features and target to
the list

all_data.append(np.append(mean_featu
res, target))

    # Define column names for the
DataFrame
    num_features =
len(mean_features)
    columns = [f'Feature_{i+1}' for
i in range(num_features)] +
['Target']

    # Create DataFrame from the list
    final_df =
pd.DataFrame(all_data,
columns=columns)


    # Save aggregated features with
targets into a single CSV file
    output_file_path =
os.path.join(big_directory,
'aggregated_features_with_targets.cs
v')

final_df.to_csv(output_file_path,
index=False)
```

8.3. Training Data Shuffling for Improved Predictive Code

```
import pandas as pd
from sklearn.model_selection import
train_test_split
from sklearn.ensemble import
RandomForestClassifier


# Read data from CSV file
df = pd.read_csv('/content/drive/My
Drive/training
data/aggregated_features_with_target
s.csv')
dataSet = df.sample(frac=1,
random_state=42) # Shuffle the data

# Split columns between features and
target
training_features =
dataSet.drop('Target', axis=1)
training_target = dataSet['Target']
```

8.4. Testing Data Reading Code

```
from sklearn.metrics import
accuracy_score,
classification_report,
confusion_matrix

dataSet =
pd.read_csv('/content/drive/My
Drive/testing
data/aggregated_features_with_target
s.csv')

# Split columns between features and
target
testing_features =
dataSet.drop('Target', axis=1)
testing_target = dataSet['Target']
```

8.5. KNN Classifier Code

```


import pandas as pd

from sklearn.ensemble import
RandomForestClassifier
from sklearn.metrics import
accuracy_score,
classification_report
from sklearn.model_selection import
train_test_split, GridSearchCV
from sklearn.neighbors import
KNeighborsClassifier

# Applying the knn Classifier With K
= 4 to the data
knnClassifierWithK4 =
KNeighborsClassifier(n_neighbors=4)
knnClassifierWithK4.fit(training_fea
tures, training_target)
```

```
# Predicting the test set results
knnClassifierWithK4Predictions =
knnClassifierWithK4.predict(testing_
features)

# Calculating the accuracy of the
model
knnClassifierWithK4Accuracy =
accuracy_score(testing_target,
knnClassifierWithK4Predictions)
print(f'KNN Classifier with K=4
Accuracy:
{knnClassifierWithK4Accuracy}')
print(classification_report(testing_
target,
knnClassifierWithK4Predictions))
# Print real and predicted labels
for i in range(len(testing_target)):
    print(f'Real:
{testing_target[i]}, Predicted:
{knnClassifierWithK4Predictions[i]}'
)
```

8.6. Optimizing SVM Classifier Code for Predictive Modeling

```
from sklearn.svm import SVC
from sklearn.model_selection import
GridSearchCV
# Support Vector Machine (SVM)
SVM = SVC()

# Hyperparameters to test
svm_parameters = {'kernel':
['linear', 'poly', 'rbf',
'sigmoid'], 'degree': [2, 3, 4, 5]}

# Grid Search with cross-validation
svm_grid_search =
GridSearchCV(estimator=SVM,
param_grid=svm_parameters, cv=5,
scoring='accuracy', n_jobs=-1)
svm_grid_search.fit(training_feature
s, training_target)

# Best parameters and score
svm_best_params =
svm_grid_search.best_params_
svm_best_score =
svm_grid_search.best_score_

print("Best Parameters for SVM:",
svm_best_params)
print("Best Cross-Validation
Score:", svm_best_score)

best_svm_model =
svm_grid_search.best_estimator_
predictedTargetSVM =
best_svm_model.predict(testing_featu
res)
accuracySVM =
accuracy_score(testing_target,
predictedTargetSVM)
print("Accuracy of SVM :",
accuracySVM)
print(classification_report(testing_
target, predictedTargetSVM))

# Print real and predicted labels
for i in range(len(testing_target)):
    print(f'Real:
{testing_target[i]}, Predicted:
{predictedTargetSVM[i]}')
```

8.7. Optimizing Random Forest Classifier for Predictive Modeling

```
# Random Forest
RFC = RandomForestClassifier()

# Hyperparameters to test
rfc_parameters = {"n_estimators":
[10, 50, 100, 200], "max_depth": [5,
10, 20, 50]}

# Grid Search with cross-validation
RFC_classifier_grid_search =
GridSearchCV(estimator=RFC,
param_grid=rfc_parameters, cv=5,
scoring='accuracy', n_jobs=-1)
```

```
RFC_classifier_grid_search.fit(train
ing_features, training_target)

# Best parameters and score
RFC_best_params =
RFC_classifier_grid_search.best_para
ms_
RFC_best_score =
RFC_classifier_grid_search.best_scor
e_

print("Best Parameters for RFC:",
RFC_best_params)
print("Best Cross-Validation
Score:", RFC_best_score)

best_rfc_model =
RFC_classifier_grid_search.best_esti
mator_
predictedTargetRFC =
best_rfc_model.predict(testing_featu
res)
accuracyRFC =
accuracy_score(testing_target,
predictedTargetRFC)

print("Accuracy of RFC :",
accuracyRFC)
print(classification_report(testing_
target, predictedTargetRFC))


# Print real and predicted labels
for i in range(len(testing_target)):
    print(f'Real:
{testing_target[i]}, Predicted:
{predictedTargetRFC[i]}')
```