California State University, Fresno
Lyles College of Engineering
Electrical and Computer Engineering Department

PROJECT REPORT

Assignment:	**PROJECT 1**
Project Title:	**Classification of Spoken Digits using PCA/SVD,SVM, Decision Trees and k-NN**
Course Title:	**ECE 172 (Fundamentals of Machine Learning)**
Instructor:	**Dr. Hovannes Kulhandjian**

Prepared by: Aryan Singh, Omer Al Sumeri

Date Submitted: 03/15/2024

INSTRUCTOR SECTION

Comments: ____

____

____

____

____

____

____

Final Grade: ____________________

TABLE OF CONTENTS

1. Objectives

The objective of this project is to understand and implement various types of Machine Learning techniques for the accurate classification of data (images or audio files). In this project the audio files containing spoken numbers from 0-9 are being classified using the methods of SVD/PCA (Singular Value Decomposition/Principal Component Analysis), SVM (Support Vector Machines), Decision Trees and k-NN (k-Nearest Neighbor).

2. Theoretical Background

This project involves the use of four classification techniques which are SVD (Singular Value Decomposition) / PCA (Principal Component Analysis), SVM (Support Vector Machines), Decision Trees and k-Nearest Neighbor. These techniques are used to provide a framework for processing and classification of different images. In this project they will be utilized to process and classify various spoken numbers from audio (.wav) files. The language used to perform this project is Python using the PyCharm IDE.

2.1 SVD/PCA for Dimensionality Reduction

SVD/PCA are techniques which are used to perform dimensionality reduction in Machine Learning.
SVD is a matrix factorization technique which breaks down a matrix into three other matrices which are orthogonal (perpendicular) to each other representing the row, column and diagonal singular values. In this project, SVD is used to reduce the dimensionality of the data by gathering the important features and discarding all the noise.
PCA is also a method of dimensionality reduction which identifies the principal components of an image in order to maximize the variation in data. The original data is then projected onto these principal components to reduce the dimensionality, while keeping as much variation as possible. In this project, PCA reduces the dimensionality of the matrices of the audio files in order to make them easier to analyze and classify.

2.2 SVM (Support Vector Machines)

SVM is a supervised learning algorithm tool that is used for regression and then classification of data. The main purpose of SVM is to find the hyperplane that best separates the different classes of data in the feature space. This method maps the input data into a high dimensional feature space and finds the best hyperplane that separates the data through multiple iterations.

In this project SVM can be used to classify the different numbers being spoken in the audio files by repetitive iteration through the given set of data. This way, it can effectively learn how to distinguish between different sets of digits.

2.3 Decision Trees

Decision trees are non parametric supervised learning algorithms used for regression and classification tasks. The decision trees partition the feature space into different regions based on the feature values, therefore creating a tree-like structure where each node represents a decision based on the features. This method regresses the data based on the true or false answers received from each node. In this project, decision trees can be used to partition the feature space according to the audio features extracted from the .wav files. Repetition of such decisions and splitting of branches creates a tree-like structure which allows classification of data (numbers in this case).

2.4 k-NN (k-Nearest Neighbor)

k-Nearest Neighbor is a supervised learning algorithm which classifies data based on the majority class of their k-Nearest neighbors in the feature space. The variable 'k' is used to determine how many neighbors are considered for the classification. This method is easy to implement and is effective since it does not make any assumptions about the distribution of the data being fed to it for training. The choice of the 'k' variable is what significantly impacts the performance of this model. In this project k-NN is used to classify the numbers based on the similarity of their audio waveform features. By calculating the distance between the features of the .wav file and their k-nearest neighbors, this method can assign class labels to each waveform sample.

3. Procedure

3.1 SVD/PCA Classification Method

The first method used for classification of various spoken digits was the PCA/SVD classification method. Before beginning the first part, all of the necessary libraries were imported into the project directory so that the functions present in them can be utilized throughout the project.
The libraries that were imported for the PCA/SVD classification were 'os' which allowed us to interact with the operating system, like opening and accessing a file. The 'librosa' library was also imported which was used for audio processing tasks such as loading audio files. The 'numpy' library was also used under the alias of 'np' in order to perform numerical computations in python. Finally, the 'matplotlib.pyplot' library was used as 'plt' to plot 3D graphs, similar to the plot functions in MATLAB.
After all necessary libraries were imported the PCA/SVD classification was initiated. The audio files were downloaded from github because all of the audio files present on github were 8Khz and did not require any resampling or synchronization (so we had no synchronization issues and we did not have to resample them to be the same frequency).
The load_audio_files_from_directory function was used to load all the audio files present in our directory which had the frequency of 8Khz, using the 'librosa.load' function. The function

returned the audio files array which contained all of the audio samples as waveforms. The load_audio_data function was used to load the audio files for all digits from 0-9 from their directories and append them into the audio_data_matrix to create a matrix.
Then in order to make all elements of the matrix the same size, 2 functions called find_max_samples and find_max_samples_position were created. The find_max_samples function calculated the largest number of samples in the audio data matrix and the find_max_samples_position function found out which element had the maximum number of samples. After finding out the largest size, the append_zeros_to_all_audio_files function was used in order to append zeros at the end of each audio file to make them the same size (the size of the largest file).
Once we had all the files with the same number of samples, the noise present in each audio file was removed using the remove_low_amplitude_segments. This was done in order to reduce noise and increase the accuracy of our system. After performing all of the matrix manipulations, the final matrix is ready for classification.
The function plot_all_classificationt_PCA is then used to plot the 3 main principal components for the classification of all 10 digits on a 3D scatter plot. The same was repeated for 2 specific digits to visualize the difference between their datasets.
After training the model to classify the numbers based on their waveforms, a random audio sample was selected and plotted on the 3D scatter plot. This file was then trimmed to eliminate noise and appended with zeros to make it the same size as the other files. Following the steps in the handout, this file was then flattened and stacked in columns of matrix B. The average of the new matrix B was calculated and then it was centered by subtracting the average from it.
SVD was performed on this file and the results were dimensionally reduced and visualized using PCA to classify the unknown audio file as one of the ten digits.
The centered matrix after calculating the SVD was saved as the projected_data to be used for the next parts of the project.

3.2 SVM Classification Method

For the SVM classification, all of the libraries included for SVD/PCA classification were included here too. The load function from 'librosa' library was used to load in the centered audio matrix from the last classification.
After the matrix had been loaded, the svm_classification_all_digits was used to perform SVM classification for all 10 digits. Using the train_test_split, all of the data was split into testing and training data. An SVM classifier with a RBF (Radial basis function) kernel was fitted for the training data and a predicted label for the test data was created. Using the testing and predicted labels, the accuracy was calculated and a classification report was generated showing the accuracy of the predicted classification made.
The function svm_classification_two_digits was then used in order to perform SVM classification on only the 2 specified digits, because the accuracy rate when using 10 digits was low and while using just 2 digits it was quite high. This function gathered the data associated

with these 2 digits and split them into training and testing data. Then a predicted label was created for the test data and the accuracy for the predicted label data and the test data was calculated. The classification report was also computed for the same. Since only 2 digits were used the accuracy came out to be higher than when 10 digits were used.
The principal components from Part A were then loaded using the load_projected_data and the labels for each of the 1000 samples were generated (100 samples of the 10 digits so in total 10*100 = 1000 labels). The SVM classification function for all digits was then used for the principal components of the matrix from part A and the 1000 labels generated for them. The result and the classification report of all the digits was printed out.
The same procedure was done but for only 2 digits this time using the svm_classification_two_digits function. The digits 0 and 4 were selected and SVM classification, of the principal components from Part A with their labels, was done. The accuracy and the classification report of these 2 digits was then printed out to see the difference in accuracy.

3.3 Decision Trees Classification Method

The Decision tree Classification Method utilized all of the libraries that were used for the previous 2 classifications. So all of those libraries were imported in the python file. One different library added here was the DecisionTreeClassifier which implements the decision tree algorithm for classification purposes.
The code for this classification method was similar to the SVM classification method, where the projected_data file was loaded from part A which had the centered audio matrix stored in it. The train_evaluate_decision_tree_all_digits function was used to split the data into training and testing data, similar to the SVM implementation. It then performs classification for all 10 digit files using DecisionTreeClassifier. The classifier is then fitted onto the training data, and then the function predicts the labels for the test data. After the labels have been created the accuracy along with the classification report are generated for the predicted and the test data.
Similarly, the function train_evaluate_decision_tree_two_digit function is used to perform the decision tree classification but for only 2 specific digits that have been passed in as arguments into the function. Once the digits have been passed in, then the function performs splitting of data into training and testing data, followed by fitting the data using the decision tree classifier. After performing the fitting for the 2 digits, the function then predicts the labels for the files of those 2 digits and computes the accuracy along with generating the classification report.
After defining the functions the projected data was loaded in using the load function and the 1000 labels for the sample data were created. This was then passed into the train_evaluate_decision_tree_all_digits function to calculate the accuracy and generate the classification report of all 10 digits. The principal components and the labels were then passed into the train_evaluate_decision_tree_two_digit function and the accuracy and classification report of the 2 digits (0 and 4 in this case too), using the principal components and the labels, were generated.

3.4 k-Nearest Neighbor Classification Method

For the k-Nearest Neighbor classification, all the same libraries were used along with one change. For the k-NN classification, the kNeighborClassifier was imported in order to implement the k-NN algorithm. The load projected data function was added from part A which contained the principal components calculated using PCA in part A.
The k-NN classifier all classes function was created which performed k-NN classification for all 10 digits using the K value of 31. The function took in the argument of the projected data (the principal components). The data was then split into training and testing data, followed by the use of the StandardScaler function to scale the features present inside the passed in matrix. The training and testing data is then fitted with the K value of 31, and the predicted labels for the scaled testing data was created. After the creation of the labels, the accuracy was computed for the predicted labels and the testing data.
Similar to the last 2 methods, this method also had the function which classified between 2 digits using the k-NN method. The 2 digit kNN classifier function was made which had the K value of 15 and performed the kNN classification for 2 specific digits. The data was filtered according to the 2 digits that were specified and then the training and testing data was split. The Scaler function was used followed by the fitting of the kNN classifier for the specific K value (15). Then the labels were predicted and the accuracy for the predicted labels and the testing data was calculated.
After the functions were defined the projected data was loaded using the load projected data function to get the principal components from part A. The labels for the 1000 samples were then generated and stored in a variable. The kNN classifier function for all 10 digits was then called to return the accuracy of kNN classification for all 10 digits.
Once that was printed, the kNN classifier function to classify 2 specific digits was called with the digits specified as 0 and 4. The accuracy of the kNN classification for those 2 digits was then printed out to see the difference.

4. Analysis and Discussion of Results

4.1 SVD/PCA Classification Method

Figure 4.1 contains the 3D model graph of the three main principle components for all 10 classifications, the spoken digits zero through nine. Each spoken digit dominates an area on the 3D plot which shows that there is clear separation between each class. Due to the number of classifications, some classifications are not able to be seen. The spoken digit zero, is in the middle which is covered by all the other digits.

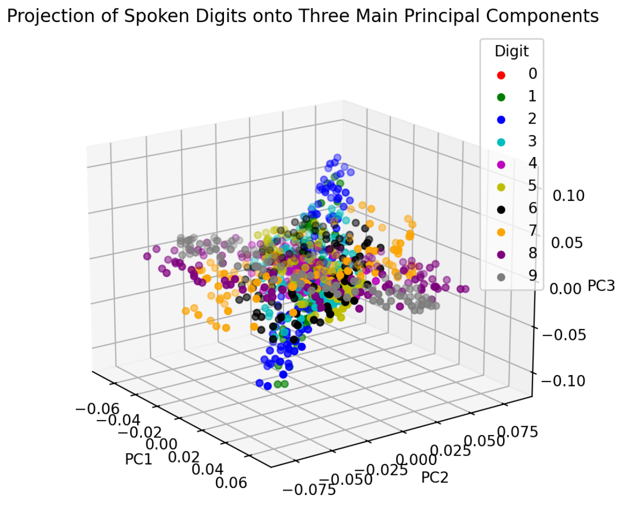


Figure 4.1. Projection of the Three Main Principal Components on a 3D graph

Figure 4.2 only shows two classifications, spoken digit zero and seven, on the 3D plot. It can be seen that digit zero is condensed in the middle, while digit seven has a ring-like structure around it. Figure 4.1 and Figure 4.2 show the three main principal components on a 3D plot, which all digits have separation to it.

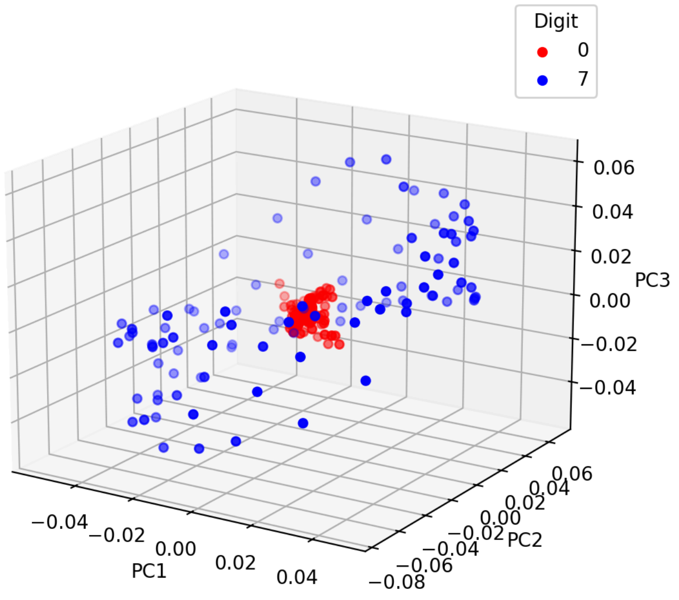


Figure 4.2. Projection of the Three Main Principal Components on a 3D graph for spoken digits 0 and 7

Figure 4.3 shows the 3D model of the three main principal components with the classification

0,7, and an “unknown audio file”. The unknown audio file is spoken digit number 7 from a different speaker than the original two speakers used in the dataset.

The process that was done to take the SVM of the unknown audio file was the same as the SVM of the audio data matrix with all the classifications. When the file was “centered”, the dimensions were off so that was not done anymore. When the unknown audio digit was graphed, the variation was large with the structure not being uniform. Figure 4.3 shows the green dots, the unknown audio file, and it crowds the other two digits, 0 and 7.

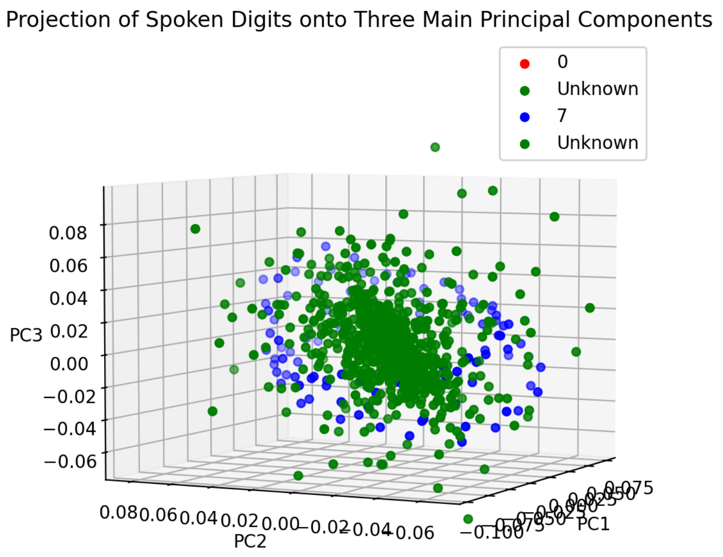
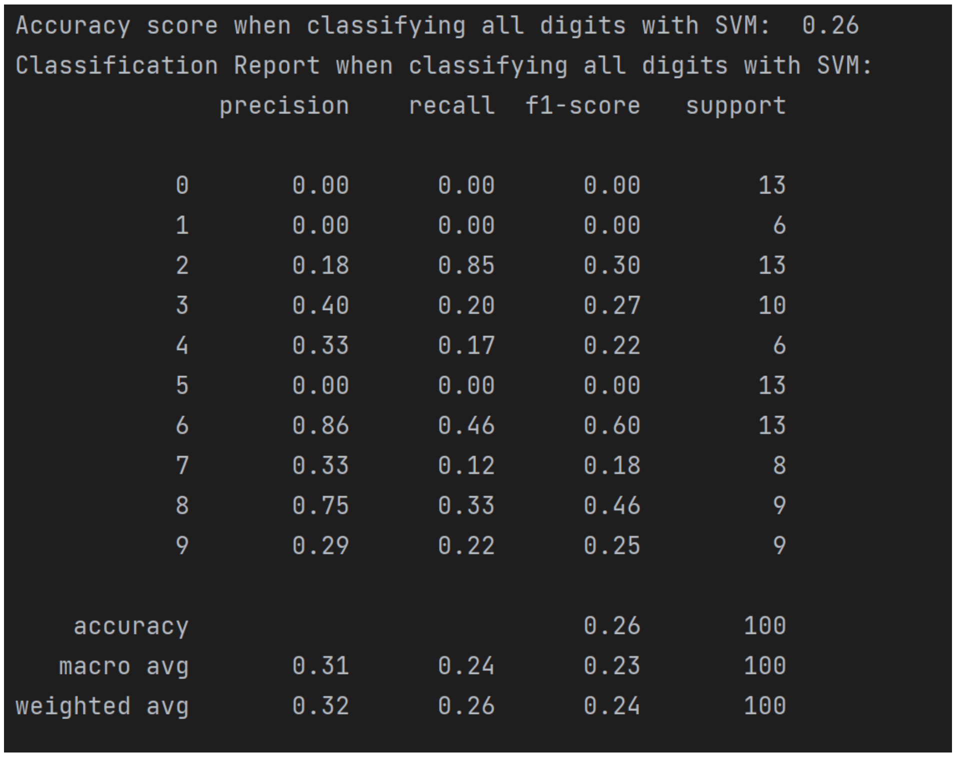


Figure 4.3. Projection of the Three Main Principal Components on a 3D graph for spoken digits 0,7 and an unknown audio files

4.2 SVM Classification Method

Figure 4.4 shows the accuracy score and classification report when classifying all the digits by SVM. The final accuracy score is 26% which is low. This can be the result of many factors the team chooses to investigate. One of the factors can be the feature extraction method used, SVD. For audio signals, it is recommended to use Mel-frequency cepstral coefficients (MFCCs) for feature extraction. Another factor can include that SVM is not the best classifier of spoken digits.

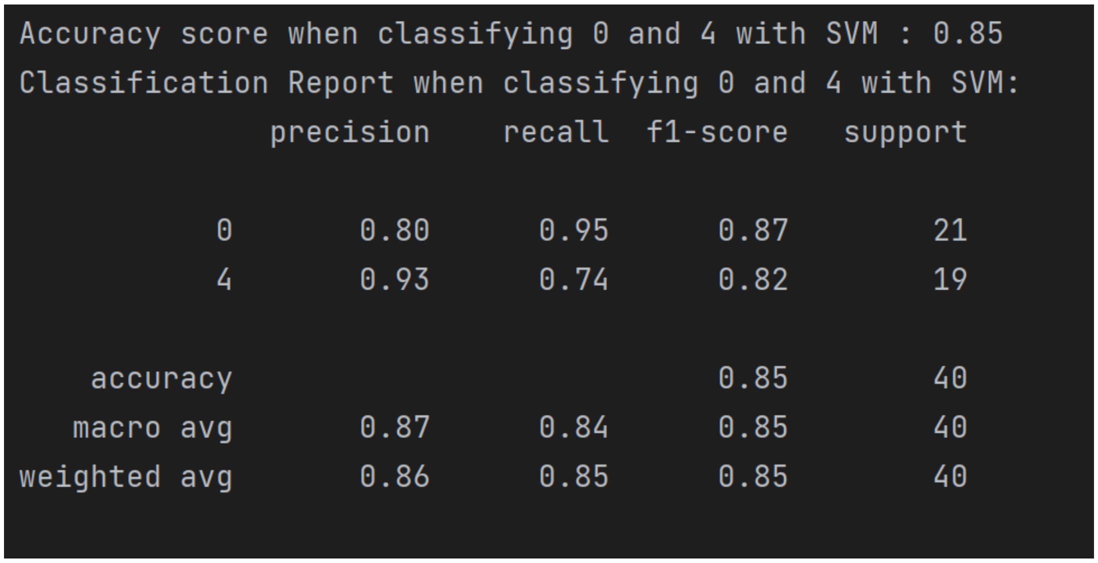
Nonetheless, the classification report gives which numbers gave the best classification. Classification 0,1 and 5 had 0% precision rate. The classification that gave the best precision rates were spoken digit 8 at 86%.



```
Accuracy score when classifying all digits with SVM:  0.26
Classification Report when classifying all digits with SVM:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        13
           1       0.00      0.00      0.00         6
           2       0.18      0.85      0.30        13
           3       0.40      0.20      0.27        10
           4       0.33      0.17      0.22         6
           5       0.00      0.00      0.00        13
           6       0.86      0.46      0.60        13
           7       0.33      0.12      0.18         8
           8       0.75      0.33      0.46         9
           9       0.29      0.22      0.25         9

    accuracy                           0.26       100
   macro avg       0.31      0.24      0.23       100
weighted avg       0.32      0.26      0.24       100
```

Figure 4.4. Accuracy and classification report when classifying all digits using SVM

Since the SVM classification method gave such low results, one factor that can give us a low accuracy rating is the number of classifications that is in the dataset. To test this, SVM was used to classify only two spoken digits, 0 and 4. Figure 4.4 contains the accuracy score and the classification report. The accuracy score when only classifying spoken digits 0 and 4 is 85%. A huge increase from 26%. Each digit also delivered high precision with 0 being classified as correct 80% of the time and 4 being classified 93% of the time.
This shows us that the SVD feature extraction method has merit and the data was not lost using the SVD method. This also shows that the number of classifications gave trouble when using the SVM model.

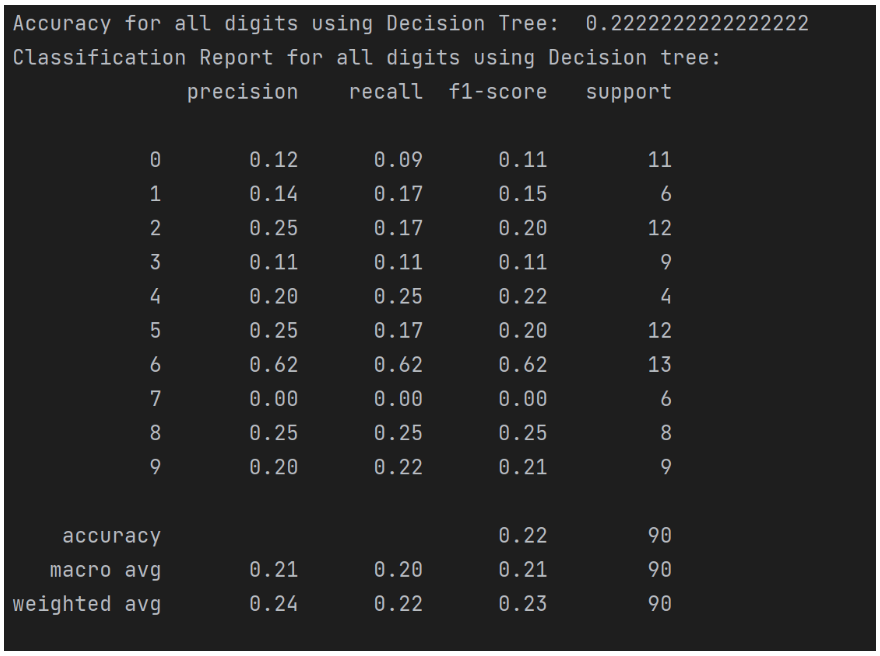


```
Accuracy score when classifying 0 and 4 with SVM : 0.85
Classification Report when classifying 0 and 4 with SVM:
              precision    recall  f1-score   support

           0       0.80      0.95      0.87        21
           4       0.93      0.74      0.82        19

    accuracy                           0.85        40
   macro avg       0.87      0.84      0.85        40
weighted avg       0.86      0.85      0.85        40
```

Figure 4.5 Accuracy and classification report when classifying spoken digits 0 and 4 using SVM

4.3 Decision Tree Classification Method

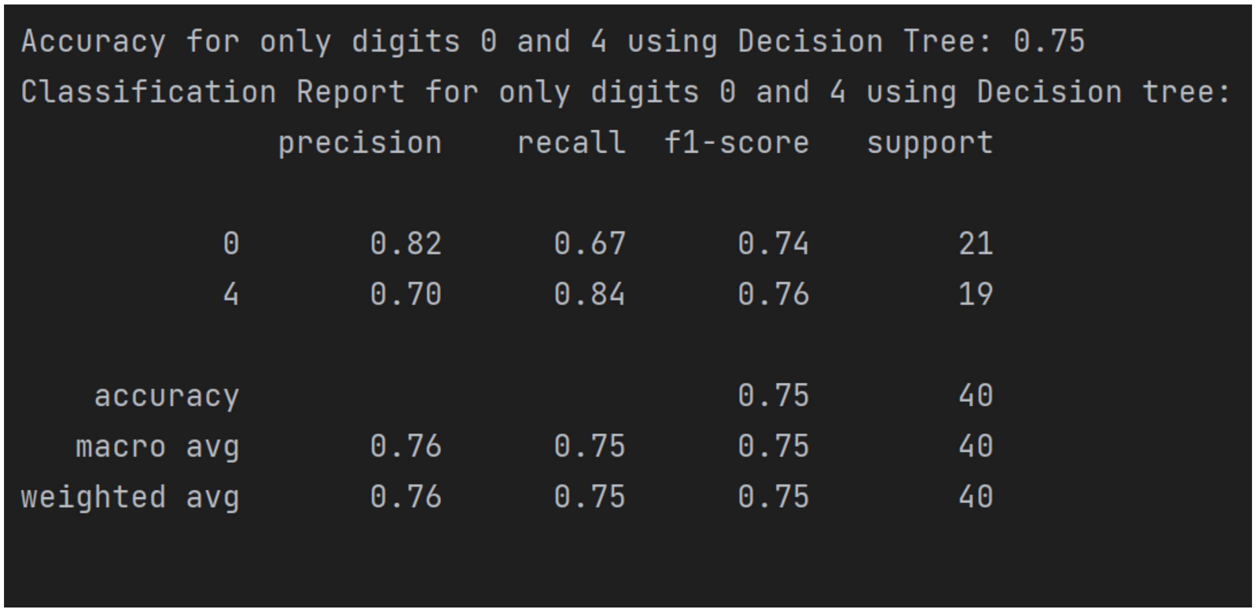
Figure 4.7 contains the accuracy result and the classification report for the spoken audio dataset using the classification method, Decision Trees. The accuracy for classifying all the spoken audio digits is 22.22%.The classification report shows that most of the classifications had horrible precision rates, with spoken digit 6 being the only one above 50%. As stated in section 4.2, there can be multiple reasons for the low accuracy rate such as the feature extraction method not being optimal for audio files or the classification method itself not being optimal.



```
Accuracy for all digits using Decision Tree:  0.2222222222222222
Classification Report for all digits using Decision tree:
              precision    recall  f1-score   support

           0       0.12      0.09      0.11        11
           1       0.14      0.17      0.15         6
           2       0.25      0.17      0.20        12
           3       0.11      0.11      0.11         9
           4       0.20      0.25      0.22         4
           5       0.25      0.17      0.20        12
           6       0.62      0.62      0.62        13
           7       0.00      0.00      0.00         6
           8       0.25      0.25      0.25         8
           9       0.20      0.22      0.21         9

    accuracy                           0.22        90
   macro avg       0.21      0.20      0.21        90
weighted avg       0.24      0.22      0.23        90
```

Figure 4.7 Accuracy and classification report when classifying all digits using Decision Trees

As done in Section 4.2, the classification method decision tree will be executed to only classify two spoken digits to see if there is a better accuracy rate with less number of classifications. Figure 4.8 has the accuracy and classification report when classifying digits 0 and 4 using decision trees. The accuracy result when using decision trees with only the spoken digits 0 and 4 is 75%. The classification report shows that the precision is 82% for 0 and 70% for 4.

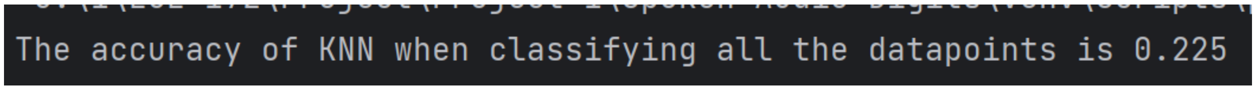


```
Accuracy for only digits 0 and 4 using Decision Tree: 0.75
Classification Report for only digits 0 and 4 using Decision tree:
              precision    recall  f1-score   support

           0       0.82      0.67      0.74        21
           4       0.70      0.84      0.76        19

    accuracy                           0.75        40
   macro avg       0.76      0.75      0.75        40
weighted avg       0.76      0.75      0.75        40
```

Figure 4.8 Accuracy and classification report when classifying only digits 0 and 4 using Decision Trees

4.4 k-Nearest Neighbor Classification Method

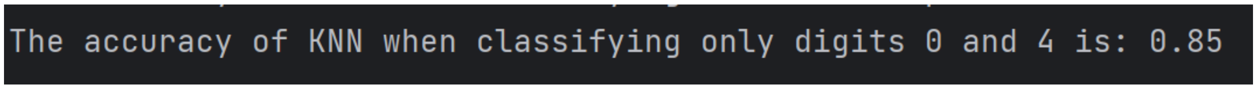
Figure 4.9 has the accuracy result for KNN when classifying all the spoken digits. When using KNN, the accuracy result is 22.5%. As stated in section 4.2 and 4.3, there can be multiple reasons for the low accuracy rate such as the feature extraction method not being optimal for audio files or the classification method itself not being optimal.



```
The accuracy of KNN when classifying all the datapoints is 0.225
```

Figure 4.9 Accuracy result when using the classifying all spoken audio digits with the classification algorithm KNN

As done in Section 4.2, the classification method decision tree will be executed to only classify two spoken digits to see if there is a better accuracy rate with less number of classifications. When using the KNN classification algorithm and classifying only spoken audio digits 0 and 4, the accuracy rate is 85%. Much higher than the accuracy result classifying all the spoken audio digits.



```
The accuracy of KNN when classifying only digits 0 and 4 is: 0.85
```

Figure 4.10 Accuracy result when using the classifying audio digit 0 and 4 with the classification algorithm KNN

4.5 Analyzing all the classification algorithms

When analyzing all the classification algorithms with the entire dataset, SVM, decision trees, and KNN, SVM gives the highest accuracy rate of 26%. The next best classification algorithm is KNN with 22.5% and the worst performing algorithm is decision trees with a rate of 22.22%.

All of these algorithms gave non optimal results which can be the result of different reasons. The reasons are the classification algorithms are not optimal for classifying speaking audio digits and the feature extraction method used, or the classification algorithm used does not handle the amount of classes the dataset has.

To test if the problem for the classification algorithms is the number of classifications in the dataset, the classifications algorithms were retested with classifying only two spoken digits, zero and four. The accuracy results when classifying only two spoken digits were much higher compared to the accuracy results for all the classification. SVM and KNN gave the same result of 85% accuracy. Decision Trees gave an accuracy rating of 75%.

5. Conclusion

In conclusion, this project was able to provide us with a great insight into how spoken digits in audio files are classified using different data classification algorithms like SVD/PCA, SVM, Decision Trees and k-Nearest Neighbor. We were able to understand the implementation of each of these algorithms along with their advantages and disadvantages. The use of these algorithms to convert audio files to matrices and then utilizing these algorithms to classify the data is something new that we learned through this project.

6. Appendix

6.1 Python Code for SVD/PCA Classification Method

```
import os
import librosa
import numpy as np
import matplotlib.pyplot as plt

# Function to load audio files from a directory
def load_audio_files_from_directory(directory):
    audio_files = []
    for filename in os.listdir(directory):
        if filename.endswith('.wav'):
            file_path = os.path.join(directory, filename)
            audio_data, _ = librosa.load(file_path, sr=8000)
            audio_files.append(audio_data)
    return audio_files


# Main function to load audio files from all directories
def load_audio_data(root_directory):
    audio_data_matrix = []
    for digit in range(10):
        digit_directory = os.path.join(root_directory, str(digit))
        digit_audio_files = load_audio_files_from_directory(digit_directory)
        audio_data_matrix.append(digit_audio_files)
    return audio_data_matrix


# Function to find the max number of samples in the audio data matrix
def find_max_samples(audio_data_matrix):
    max_samples = 0
    for i in range(10):
        for j in range(100):
            if len(audio_data_matrix[i][j]) > max_samples:
                max_samples = len(audio_data_matrix[i][j])
    return max_samples


# Function to find the which audio file has the max amount of samples
def find_max_samples_position(audio_data_matrix):
    digit_max = 0
    audio_file_num = 0
    max_samples = 0
    for i in range(10):
        for j in range(100):
            if len(audio_data_matrix[i][j]) > max_samples:
                max_samples = len(audio_data_matrix[i][j])
                digit_max = i
                audio_file_num = j
    return digit_max, audio_file_num


# Function to append zeros to all the audio files for them to be the same size
def append_zeros_to_all_audio_files(audio_data_matrix):
    for i in range(10):
        for j in range(100):
            if len(audio_data_matrix[i][j]) < max_samples:
                audio_data_matrix[i][j] = np.append(audio_data_matrix[i][j],
                                                    [0] * (max_samples - 
len(audio_data_matrix[i][j])))
    return audio_data_matrix


# Function to remove amplitudes that are not within a certain threshold
def remove_low_amplitude_segments(audio_data_matrix, threshold=0.02):
    trimmed_audio_data_matrix = []
    for digit_audio_files in audio_data_matrix:
        trimmed_digit_audio_files = []
        for audio_data in digit_audio_files:
            # Find segments with amplitude above the threshold
            non_zero_indices = np.where(np.abs(audio_data) > threshold)[0]

            # Get the start and end indices of non-zero segments
            if len(non_zero_indices) > 0:
                start_idx = non_zero_indices[0]
                end_idx = non_zero_indices[-1]

                # Extract the non-zero segment from the signal
                trimmed_audio = audio_data[start_idx:end_idx + 1]
                trimmed_digit_audio_files.append(trimmed_audio)
        trimmed_audio_data_matrix.append(trimmed_digit_audio_files)
    return trimmed_audio_data_matrix

# Function to remove amplitudes from a single audio file
# that are not within a certain threshold

def remove_low_amplitude_segments_single(audio_data, threshold=0.02):
    # Find segments with amplitude above the threshold
    non_zero_indices = np.where(np.abs(audio_data) > threshold)[0]

    # Get the start and end indices of non-zero segments
    if len(non_zero_indices) > 0:
        start_idx = non_zero_indices[0]
        end_idx = non_zero_indices[-1]

        # Extract the non-zero segment from the signal
        trimmed_audio = audio_data[start_idx:end_idx + 1]
        return trimmed_audio
    else:
        # If no segment found above the threshold, return the original audio data
        return audio_data

# Function to 3D plot the 3 main principal components for all 10 classifications
def plot_all_classificationt_PCA(projected_data):
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')

    colors = ['r', 'g', 'b', 'c', 'm', 'y', 'k', 'orange', 'purple', 'gray']

    # Scatter plot for each spoken digit
    for digit, color in zip(range(10), colors):
        digit_indices = range(digit * 100, (digit + 1) * 100)
        ax.scatter(projected_data[digit_indices, 0], projected_data[digit_indices, 1], 
projected_data[digit_indices, 2],
                   c=color, label=str(digit))

    ax.set_xlabel('PC1')
    ax.set_ylabel('PC2')
    ax.set_zlabel('PC3')
    ax.set_title('Projection of Spoken Digits onto Three Main Principal Components')
    ax.legend(title='Digit')
    plt.show()

# Function to 3D plot the 3 main principal components for only two selected digits


def plot_classification_PCA_two_digits(projected_data, digit1, digit2):
    # Create a figure and a 3D axis
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')

    # Define colors for the two digits
    colors = ['r', 'b']

    # Scatter plot for each specified digit
    for digit, color in zip([digit1, digit2], colors):
        digit_indices = range(digit * 100, (digit + 1) * 100)
        ax.scatter(projected_data[digit_indices, 0], projected_data[digit_indices, 1],
projected_data[digit_indices, 2],
                   c=color, label=str(digit))

    # Set labels and title
    ax.set_xlabel('PC1')
    ax.set_ylabel('PC2')
    ax.set_zlabel('PC3')
    ax.set_title('Projection of Spoken Digits onto Three Main Principal Components')

    # Add legend
    ax.legend(title='Digit')

    # Show plot
    plt.show()

def plot_classification_PCA_two_digits_unknown(projected_data, digit1, digit2,
unknown_projected_data):
    # Create a figure and a 3D axis
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')

    colors = ['r', 'b']
    unknown_color = 'g'

    for digit, color in zip([digit1, digit2], colors):
        digit_indices = range(digit * 100, (digit + 1) * 100)
        ax.scatter(projected_data[digit_indices, 0], projected_data[digit_indices, 1],
projected_data[digit_indices, 2],
                   c=color, label=str(digit))


        ax.scatter(unknown_projected_data[0], unknown_projected_data[1],
unknown_projected_data[2],
                   c=unknown_color, label='Unknown')

    # Set labels and title
    ax.set_xlabel('PC1')
    ax.set_ylabel('PC2')
    ax.set_zlabel('PC3')
    ax.set_title('Projection of Spoken Digits onto Three Main Principal Components')

    # Add legend
    ax.legend()

    # Show plot
    plt.show()
```

```
def plot_two_selected_digits(digit1, digit2, projected_data):
    indices_digit1 = range(digit1 * 100, (digit1 + 1) * 100)
    indices_digit2 = range(digit2 * 100, (digit2 + 1) * 100)

    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')

    ax.scatter(projected_data[indices_digit1, 0], projected_data[indices_digit1, 1],
projected_data[indices_digit1, 2],
               c='r', label=str(digit1))
    ax.scatter(projected_data[indices_digit2, 0], projected_data[indices_digit2, 1],
projected_data[indices_digit2, 2],
               c='b', label=str(digit2))

    ax.set_xlabel('PC1')
    ax.set_ylabel('PC2')
    ax.set_zlabel('PC3')
    ax.set_title('Projection of Selected Spoken Digits onto Three Main Principal
Components')
    ax.legend(title='Digit')
    plt.show()

# Function to plot an audio signal given a spoken digit and file number
def plot_audio_signal(audio_data_matrix, digit, file_number):
    if digit < 0 or digit > 9:
        print("Invalid digit. Digit should be between 0 and 9.")
        return
    if file_number < 0 or file_number >= len(audio_data_matrix[digit]):
        print("Invalid file number for the given digit.")
        return

    audio_signal = audio_data_matrix[digit][file_number]
    plt.figure(figsize=(10, 4))
    plt.plot(audio_signal)
    plt.title(f"Digit {digit}, File {file_number}")
    plt.xlabel("Sample")
    plt.ylabel("Amplitude")
    plt.grid(True)
    plt.show()

# Function to preprocess the new audio file for part 10


# Function to 3D plot the 3 main principal components for only two selected digits
plus the new audio file


'''
    Project is to classify spoken audio digits. In my directory, I have 10 folders
    that represent the audio digits 0-9. Each folder contains 100 .wav files for
    each folder.

'''

'''
    Step 1 & 2
'''
root_directory = 'C:\\1\\ECE 172\\Project\\Data\\Spoken Digits'
audio_data_matrix = load_audio_data(root_directory)

# audio_data_matrix is a 3D numpy array where the first dimension represents digits 0
through 9,
# the second dimension represents the individual audio files, and the third dimension
contains the audio data itself.
# For example, to access the audio data for digit 0 and the 5th audio file, you can
use:
# audio_data_digit0_file5 = audio_data_matrix[0][4]


'''
#Code to verify the size of the matrix
The matrix should be

rows = len(audio_data_matrix)  # returns the amount of rows in our case (spoken
digits) should be 10

cols = len(audio_data_matrix[0])  # return the amount of audio files of the digit in
the brackets
#should be 100

length_audio_file_0_0 = len(audio_data_matrix[0][0])  # returns the length of the
audio file at first [spoken digit] and
# [row number]

print("Amount of rows in the matrix (Spoken digits): {}" .format(rows))
print("Amount of audio files for digit 0: {}".format(cols))
print("The number of samples for the first audio file for digit 0:
{}".format(length_audio_file_0_0))

'''

'''
    Step 3.a

    For since I am using audio files, I had to do additional steps.

    After completing all of part A, I noticed that all the digits in the
    PCA plot was not distinguishable. I believe its because the all of the zeros that
I am appending
    to the other audio files.

'''
trimmed_audio_data = remove_low_amplitude_segments(audio_data_matrix)

'''
    Step 3.b
    All audio files need to have the same number of samples
    To do this, the maximum number of samples will be found in
    the Audio data matrix, and that will minimum number of samples
    for all audio files. All other files will have 0 appended to them
'''

max_samples = find_max_samples(trimmed_audio_data)

max_digit, audio_num_max = find_max_samples_position(trimmed_audio_data)

audio_data_matrix = append_zeros_to_all_audio_files(trimmed_audio_data)

'''
    Step 4

    The next part turns a 3D matrix into a 2D
    Flatten each audio file in the audio_data_matrix and collect the flattened arrays
into a list.
    Nested loop iterates over each digit and each audio file within that digit,
    flattens each audio file into a 1D array, and appends it to the
flattened_audio_data list.
    Matrix B is created by column stacking the matrix

    The rows represent the samples of each audio file with the size of max samples,
    col0 is the first sample of the audio digit

    The columns represent each audio file, first 100 columns is audio digit 0, total
is 1000
    final matrix size is max samples x 1000
'''

flattened_audio_data = []
for digit in range(10):
    for audio_file in trimmed_audio_data[digit]:
        flattened_audio_data.append(audio_file.flatten())

# Stack Flattened Audio Files as Columns of Matrix B
matrix_B = np.column_stack(flattened_audio_data)

'''
    Step 5
    I interpreted this as I need to calculate the average audio file with respect to
all 1000 audio files
    so I calculate the average amplitude of all files in sample 0 (also meaning row 0)
'''
average_audio = np.mean(matrix_B, axis=1)

'''
    Step 6
    Subtract the average audio from the original matrixB
'''
centered_matrix = matrix_B - average_audio[:, np.newaxis]

'''
    Step 7
    SVD analysis on the centered audio dataset
    # Perform Singular Value Decomposition (SVD) on matrix B
'''
# Perform Singular Value Decomposition (SVD) on matrix B
U, S, Vt = np.linalg.svd(centered_matrix,full_matrices= False)

'''

    Step 8
    Compute SVD analysis on each category


'''
# Step 8: Compute SVD analysis on each category
SVD_results = []

for digit_audio_data in trimmed_audio_data:
    flattened_digit_audio_data = [audio_file.flatten() for audio_file in
digit_audio_data]

    matrix_C = np.column_stack(flattened_digit_audio_data)

    centered_matrix_digit = matrix_C - np.mean(matrix_C, axis=1)[:, np.newaxis]

    U_digit, S_digit, Vt_digit = np.linalg.svd(centered_matrix_digit,
full_matrices=False)
```

```
    SVD_results.append((U_digit, S_digit, Vt_digit))



'''
    Step 9

    Since it is difficult to see separation with all 10 classifications
    on the 3D plot, a function is added where only two
    spoken digits are plotted

'''


plot_all_classificationt_PCA(U)
plot_classification_PCA_two_digits(U,0,7)


'''
    Step 10
    I am loading in an audio files has the spoken digit four

'''

unknown_audio_data, _ = librosa.load('0_nicolas_10.wav', sr=8000)

unknown_audio_data_trimmed =
remove_low_amplitude_segments_single(unknown_audio_data,0.02)

if len(unknown_audio_data_trimmed) < max_samples:
    unknown_audio_data_trimmed = np.append(unknown_audio_data_trimmed, [0] *
(max_samples - len(unknown_audio_data_trimmed)))

flattened_unknown_audio_data = unknown_audio_data_trimmed.flatten()

unknown_audio_data_col_stack = np.column_stack(flattened_unknown_audio_data)
centered_unknown_audio = unknown_audio_data_col_stack - average_audio[:, np.newaxis]

U_unknown, S_unknown, Vt_unknown = np.linalg.svd(centered_matrix,full_matrices= False)

plot_classification_PCA_two_digits_unknown(U,0,7,U_unknown)



'''
    Will be used to load into Part B, Parc C, and Part D
'''
projected_data = centered_matrix.T @ U[:, :3]
np.save('projected_data.npy', projected_data)
```

6.2 Python Code for SVM Classification Method

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score

# Function to load the projected data matrix from Part A
def load_projected_data():
    projected_data = np.load('projected_data.npy')
    return projected_data
```

```
def load_matrix_U_data():
    matirx_U = np.load('matrix_U.npy')
    return matirx_U
# Function to do SVM for all 10 digits
def svm_classification_all_digits(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, 
random_state=42)

    clf = SVC(kernel='rbf')
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)

    return accuracy, report

# Function to do SVM for only 2 digits
def svm_classification_two_digits(features, labels, digit1, digit2):
    # Select data corresponding to the two digits
    mask = (labels == digit1) | (labels == digit2)
    selected_features = features[mask]
    selected_labels = labels[mask]

    X_train, X_test, y_train, y_test = train_test_split(selected_features, 
selected_labels, test_size=0.2,
                                                        random_state=42)

    svm_classifier = SVC(kernel='rbf')
    svm_classifier.fit(X_train, y_train)

    y_pred = svm_classifier.predict(X_test)
    report = classification_report(y_test, y_pred)

    accuracy = accuracy_score(y_test, y_pred)

    return accuracy, report


X = load_projected_data() # the principal components calculated from Part A
y = np.repeat(np.arange(10), 100)  # the labels for the 1000 samples (10 
classifications * 100 samples)
'''
    We noticed an abnormally low accuracy rate when classifying all spoken digits. We 
also included doing SVM
    on just two spoken digits to see if the accuracy rate increased which it did.

'''
accuracy_score_all, report_all = svm_classification_all_digits(X,y)

print("Accuracy score when classifying all digits with SVM: ", accuracy_score_all)
print("Classification Report when classifying all digits with SVM:")
print(report_all)

digit_1 = 0
digit_2 = 4
accuracy_score_two_digits, report_two_digits = 
svm_classification_two_digits(X,y,digit_1,digit_2)

print("Accuracy score when classifying {} and {} with SVM : 
{}".format(digit_1,digit_2,accuracy_score_two_digits))
```

```
print("Classification Report when classifying {} and {} with
SVM:".format(digit_1,digit_2))
print(report_two_digits)
```

6.3 Python Code for Decision Trees Classification Method

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
import numpy as np

# Function to load the projected data matrix from Part A
def load_projected_data():
    projected_data = np.load('projected_data.npy')
    return projected_data

# Function to classify using decision tree for all the digits
def train_evaluate_decision_tree_all_digits(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.09,
random_state=42)


    clf = DecisionTreeClassifier()
    clf.fit(X_train, y_train)


    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    report =(classification_report(y_test, y_pred))

    return accuracy, report

# Function to classify using decision tree for only two digits

def train_evaluate_decision_tree_two_digits(X, y, digit1, digit2):
    # Filter the dataset to only include the specified two digits
    mask = (y == digit1) | (y == digit2)
    X_filtered = X[mask]
    y_filtered = y[mask]

    X_train, X_test, y_train, y_test = train_test_split(X_filtered, y_filtered,
test_size=0.2, random_state=42)

    clf = DecisionTreeClassifier()
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)

    return accuracy, report

'''
    We noticed an abnormally low accuracy rate when classifying all spoken digits. We
also included doing decision tree
    on just two spoken digits to see if the accuracy rate increased which it did.

'''
X = load_projected_data()  # the principal components calculated from Part A
```

```
y = np.repeat(np.arange(10), 100)  # the labels for the 1000 samples (10
classifications * 100 samples)

accuracy_score_all_digits, report_all_digits =
train_evaluate_decision_tree_all_digits(X,y)
print("Accuracy for all digits using Decision Tree: ", accuracy_score_all_digits)
print("Classification Report for all digits using Decision tree:")
print(report_all_digits)

digit_1 = 0
digit_2 = 4
accuracy_score_two_digits, report_two_digits =
train_evaluate_decision_tree_two_digits(X,y,digit_1,digit_2)
print("Accuracy for only digits {} and {} using Decision Tree:
{}".format(digit_1,digit_2,accuracy_score_two_digits))
print("Classification Report for only digits {} and {} using Decision
tree:".format(digit_1,digit_2))
print(report_two_digits)
```

6.4 Python Code for k-Nearest Neighbor Classification Method

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import numpy as np


# Function to load the projected data matrix from Part A
def load_projected_data():
    projected_data = np.load('projected_data.npy')
    return projected_data


# Function to use KNN for all 10 classifications, k-value is 31 since sqrt(1000) ~ 31
# sqrt(n), n being total number of datapoints for the k value is the base standard of
KNN
def knn_classifier_all_classes(projected_data, y, k=31):

    X_train, X_test, y_train, y_test = train_test_split(projected_data, y,
test_size=0.08, random_state=42)
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)

    predicted_labels = knn.predict(X_test_scaled)

    accuracy = accuracy_score(y_test, predicted_labels)

    return accuracy


# Function to use KNN for all only two digits classifications,
# k-value is 31 since sqrt(200) ~ 15
def knn_classifier_two_digits(projected_data, y, digit1, digit2, k=15):

    indices_digit1 = np.where(y == digit1)[0]
    indices_digit2 = np.where(y == digit2)[0]
    selected_indices = np.concatenate([indices_digit1, indices_digit2])

    X_selected = projected_data[selected_indices]
    y_selected = y[selected_indices]

    X_train, X_test, y_train, y_test = train_test_split(X_selected, y_selected,
test_size=0.2, random_state=42)

    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)

    predicted_labels = knn.predict(X_test_scaled)

    accuracy = accuracy_score(y_test, predicted_labels)

    return accuracy


projected_data = load_projected_data()
y = np.repeat(np.arange(10), 100)

accuracy_score_all_classes = knn_classifier_all_classes(projected_data, y)

print("The accuracy of KNN when classifying all the datapoints is",
accuracy_score_all_classes)

digit1 = 0
digit2 = 4

accuracy_score_two_digits = knn_classifier_two_digits(projected_data, y, digit1,
digit2)

print("The accuracy of KNN when classifying only digits {} and {} is:
{}".format(digit1, digit2,

accuracy_score_two_digits))
```

6.5 Link to the Dataset Used

https://github.com/Jakobovski/free-spoken-digit-dataset/tree/master/recordings