ELEC 390
Project Report

Group 33
Lauren Steel, Jacob Schaffer, Ethan Silver
20218337, 20222125, 10216012
19les5@queensu.ca, 19jsbs@queensu.ca, 19es25@queensu.ca
Friday April 14, 2023

Introduction

This report outlines the development steps taken to create a simple AI system which was then deployed in a desktop app. The process is broken down into seven steps: data collection, storage, visualization, pre-processing, feature extraction, training the classifier, and implementing the model created in a desktop app. Each phase plays a significant and necessary role in the development of a reliable and consistent AI product. Data collection is the process of collecting and measuring accurate data from several sources. Data storage is when sets are organized in a manner which is easy and convenient for use. Data visualization allows for trouble shooting with data early in the design process. Data pre-processing is the concept of fixing minor or major issues with data, to ensure that sets are usable. Feature extraction is the process of identifying and isolating various 'features' that contain key information, by utilizing these values the whole signal is not required. Using the features from the preprocessed training set, a logistic regression model is trained to classify the data into 'walking' or 'jumping' classes. Finally, the model could be deployed to a desktop application through use of the Tkinter library. Together these steps lead to the creation of a final AI product which can differentiate between motions of walking and jumping.

Data Collection

The first step in the development process is to collect any necessary data. Data collection is the process in which one collects and measures accurate data from multiple relevant sources. In the case of this design challenge, the goal is to develop an application that can distinguish between walking and jumping. The particular purpose the data serves is to act as representation of what a typical walking motion looks like in comparison to that of jumping. The data to be collected can be considered what is called free-living data. This type of data is recorded under real conditions without intervention to alter the environment in which the data is being recorded. In general, it is more challenging to process and extract information from a free-living dataset [1]. The reason being that they usually contain noise and are collected with a high degree of freedom . Handling these additional difficulties is addressed later in the design process with preprocessing filters.

To accurately obtain datasets representing both walking and jumping, an app called Phyphox was used to collect accelerometer data from each member's smartphone. An accelerometer monitors acceleration in m/s^2 (or g) along three axes, x, y, and z. Figure 1 demonstrates the Phyphox interface when a 5 second accelerometer sample is produced, graphs correlating to all three axes are developed. It should be noted that an accelerometer is very sensitive to vibrations and movement, noise and artifacts are likely to be picked up [1].

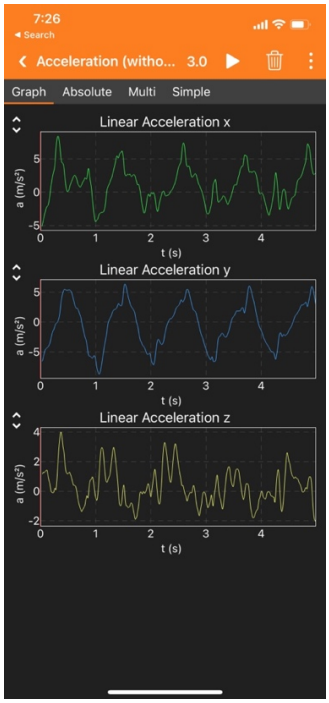


Figure 1: Phyphox interface for a 5 second accelerometer sample displaying graphs correlating to all three axes.

During the collection phase a crucial aspect to consider is diversity. A diverse dataset will allow a system to be more successful when fully developed. To maximize diversity all members of the design team collected their own set of data which would demonstrate their unique walking stride and typical jump. Moreover, each member was to ensure that throughout collection of their dataset their phone would be placed in many different positions. The variety of positions included all pant pockets, front and back, several jacket pocket samples, as well as directly carrying the phone in hand while executing the desired motions. With that being said it was ensured that the datasets collected were relatively balanced. The amount of time collected from each user, for each motion of walking or jumping, and each position would be roughly the same. The goal was to obtain a duration of data exceeding five minutes between both walking and jumping motions. The approximate outline for collection was that members would obtain thirty seconds of data for positioning in, front pockets, back pockets, jacket pockets, and being held in hand for both jumping and walking samples. Resulting in approximately two and a half minutes for each walking and jumping.

Data samples collected from the app Phyphox could be directly exported as CSV files. Therefore, data could be easily moved in any fashion most convenient to the user (email, airdrop, etc.). The transferred zip files included the CSV along with correlating meta data. Once these files for all collected samples were transferred to a member's PC it was possible to label the data as fit. Each file specified the movement (walking or jumping), position (front pocket, back pocket, jacket pocket or hand) and side (left or right) that the corresponding data represented.

Two members collected the data in thirty second segments. One member chose to collect data directly in the necessary five second samples. Those who collected samples of longer segments took different approaches in splitting the data. One member manually broke up the CSVs into

five second periods according to time stamps. The other approach used was to directly add the larger CSV files containing thirty seconds of walking or jumping acceleration data into a python file. Within the program it is possible to break up the data as shown in Figure 2.

```
Q5 = jump5
B5 = jump5.iloc[:, 0]
H51 = Q5[B5 < 5]
H52 = Q5[(5 < B5) & (B5 < 10)]
H53 = Q5[(10 < B5) & (B5 < 15)]
H54 = Q5[(15 < B5) & (B5 < 20)]
H55 = Q5[(20 < B5) & (B5 < 25)]
H56 = Q5[25 <= B5]

with h5py.File('./data.h5', 'w') as hdf:
    G1 = hdf.create_group('/Ethan')
    W = G1.create_group('/Walk')
    W.create_dataset('walk1-1', data=Z11)
    W.create_dataset('walk1-2', data=Z12)
    W.create_dataset('walk1-3', data=Z13)
    W.create_dataset('walk1-4', data=Z14)
    W.create_dataset('walk1-5', data=Z15)
    W.create_dataset('walk1-6', data=Z16)
    W.create_dataset('walk2-1', data=Z21)
```

Figure 2: A segment illustrative of the CSV to HDF5 conversion program.

Data Storage

Once data has been collected, it must be stored. Data storage should be organized in a manner which is easy and convenient for usage. After transferring all the subsets of data to a personal computer and adding labels, each member stored their collected data in the form of a HDF5 file. HDF5 is a data model as well as a library, and file format for the storage and management of data. HDF5 files are designed to handle large, complex data with an open format, meaning they are supported by many tools such as Python [2].

An HDF5 file can be viewed as a container or grouping, which holds datasets. The datasets could be a range of various file types. The two important objects in HDF5 files are ‘groups’ and ‘datasets’ [2]. Groups are like folders implemented for organization. Every HDF5 file includes a group called ‘root’. Dealing with groups and their content is very similar to path directories. A ‘/’ represents the root group and different members will follow the backslash to branch off into subgroups.

The complete HDF5 file was organized in a tree with four main groups. Three of these groups would correspond to each member’s personal datasets. The final group of the requested file is labeled ‘dataset’ and has two subgroup members corresponding to either train or test data. Creating train and test splits is vital to create a simple AI system [2]. After the team ensured each signal of their personal data was in the form of five second windows, as described in the data collection section, the data was then shuffled. 90% of collected data is used for training purposes and the remaining 10% is added to the testing subgroup. So, within the ‘dataset’ group, the contents of each subgroup train and test were taken directly from all the individual’s personally collected files but reorganized and stored with a different structure. It is typically good practice to keep the data as originally collected, which is why the groups corresponding to team members remain in the final HDF5 file. A visual of the final data organization within the HDF5 file is represented in Figure 3.

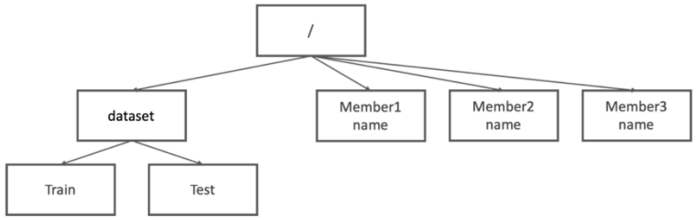


Figure 3: Final HDF5 file hierarchy visualization, Image from [3].

Each individual team member created their own HDF5 file to store their data in which they were able to organize sets as they desired. These files would likely take the form of two groups 'walking' and 'jumping' which could then branch down to various subgroups of phone positions collected (pant, jacket, hand) and the side it was taken (right, left). To start, we first include the necessary library h5py, then begin creating and writing to the HDF5 file. To ensure the proper organization it is necessary to use 'hdf.create_group' within python. This allows us to begin a hierarchy of groups.

To replicate the desired structure seen above, a separate python file is created to read induvial team members HDF5 files into one large hierarchy. The implementation code written to accomplish this can be seen in Figure 4 and Figure 5. Where the files 'ethan.h5', 'Jacob_Schaffer.h5' and 'lauren_data.h5' correspond to individual's collected data which are represented as Member1 name, Member2 name etc. in Figure 3.

```python
import h5py

ethan = h5py.File('ethan.h5', 'r')
jacob = h5py.File('Jacob_Schaffer.h5', 'r')
lauren = h5py.File('lauren_data.h5', 'r')
traintest = h5py.File('train_test.h5', 'r')
dataout = h5py.File('data.h5', 'w')

dataout.create_group('Ethan')
GEJ = dataout.create_group('Ethan/Jump')
GEW = dataout.create_group('Ethan/Walk')
dataout.create_group('Lauren')
GLJ = dataout.create_group('Lauren/Jump')
GLW = dataout.create_group('Lauren/Walk')
dataout.create_group('Jacob')
GJJ = dataout.create_group('Jacob/Jump')
GJW = dataout.create_group('Jacob/Walk')
dataout.create_group('dataset')
GTrain = dataout.create_group('dataset/Train')
GTest = dataout.create_group('dataset/Test')

# Add Ethan Data
for dataset in ethan['Jump'].keys():
    GEJ.create_dataset(name=dataset, data=ethan['Jump'][dataset])
for dataset in ethan['Walk'].keys():
    GEW.create_dataset(name=dataset, data=ethan['Walk'][dataset])

# Add Lauren Data
for dataset in lauren.keys():
    if 'jumping' in dataset:
        GLJ.create_dataset(name=dataset, data=lauren[dataset])
    if 'walking' in dataset:
        GLW.create_dataset(name=dataset, data=lauren[dataset])

```

Figure 4: Part 1 of Python code from h5_generator.py which creates an HDF5 file with desired organization.

```
# Add Jacob Data
for dataset in jacob['Jumping'].keys():
    GJJ.create_dataset(name=dataset, data=jacob['Jumping'][dataset])
for dataset in jacob['Walking'].keys():
    GJW.create_dataset(name=dataset, data=jacob['Walking'][dataset])

# Add Train Data
for dataset in traintest['dataset']['Train'].keys():
    GTrain.create_dataset(name=dataset, data=traintest['dataset']['Train'][dataset])

# Add Test Data
for dataset in traintest['dataset']['Test'].keys():
    GTest.create_dataset(name=dataset, data=traintest['dataset']['Test'][dataset])
```

Figure 5: Part 2 of Python code from file h5_generator.py which creates an HDF5 file with desired organization.

Visualization

Data visualization allows for trouble shooting with data early in the design process. It is a critical step in the field of data science, and it is a means of finding issues as well as becoming more familiar with the data. Overall, visualization helps to gain insights, identify patterns, trends, outliers, and summarize information in bigger data sets [4]. In this stage, it is necessary to visualize a few samples from the datasets on all three axes and from both motions of walking and jumping. Data visualization will be achieved using Python library Matplotlib. As shown in the plots below, there are many features that create a distinction between walking and jumping data, and these differences are taken advantage of during feature extraction.

Handheld Data

The below figures illustrate the difference in results between jumping and walking with the phone held in the user's hand. As the figures below clearly demonstrate, the data for jumping and walking are very distinct. Jumping data has much higher positive extremes on all axes, and much lower negative extremes on the Y axis. The data also simply deviates much more, which can be utilized during feature extraction.

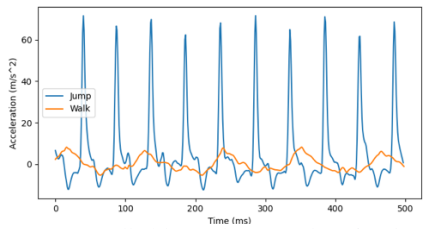


Figure 6: Handheld Acceleration data for the X axis.

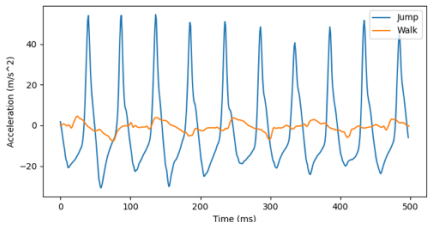


Figure 7: Handheld Acceleration data for the Y axis.

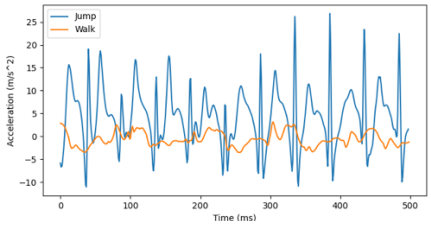


Figure 8: Handheld Acceleration data for the Z axis.

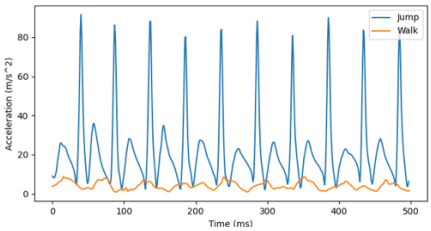


Figure 9: Handheld Absolute Acceleration data.

Pocket Data

Below are figures which contrast the accelerometer data collected while the user has their phone in their pocket. The data is much noisier and less consistent than with the phone in the user's hand, which makes sense since the phone will naturally shake around more in a pocket than in a hand. While the distinct features of the data are less obvious than with the phone in the user's hand, jumping data still has distinct features from walking data, particularly on the Y axis, as shown in Figure 11. Acceleration values reach much more extreme negatives for walking than for jumping. Another field that still expresses very distinct values is the absolute acceleration data, shown in Figure 13.

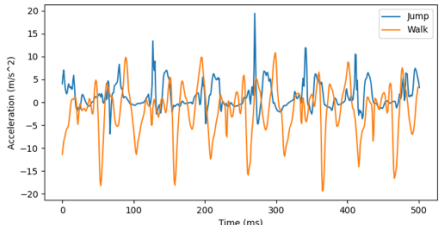


Figure 10: Pocket Acceleration data for the X axis.

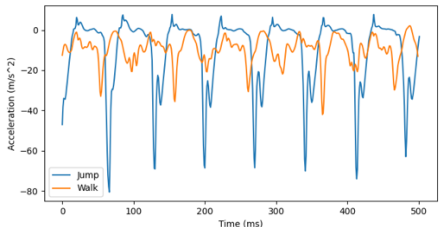


Figure 11: Pocket Acceleration data for the Y axis.

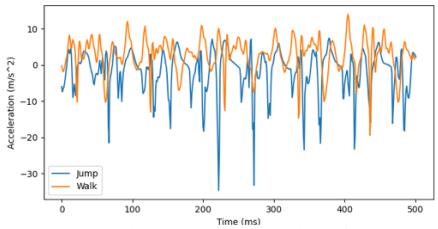


Figure 12: Pocket Acceleration data for the Z axis.

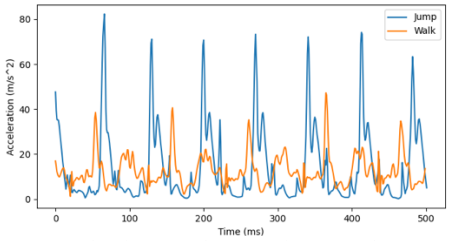


Figure 13:Absolute Pocket Acceleration data.

Future Considerations

While most of the data collected turned out clean as shown above, as it was collected in a controlled environment, real-world data may have significantly more noise. The following data was some of the noisier data collected, and if the model were to be redeveloped with more robust functionality, the variety of real-world results compared to controlled data collection would be a more important consideration.

As shown below, the differences between jumping and walking acceleration are much less pronounced, though there are still distinct features that allow it to be classified. The unique features selected and extracted will be discussed further in Pre-processing.

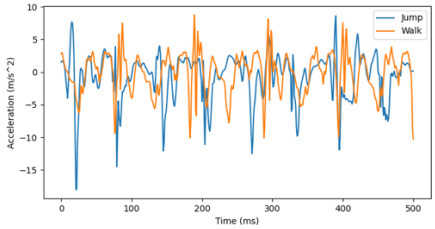


Figure 14: Real-world Pocket Acceleration for the X Axis.

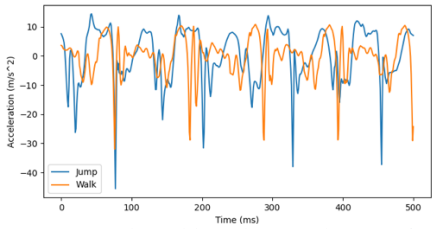


Figure 15: Real-world Pocket Acceleration for the Y Axis.

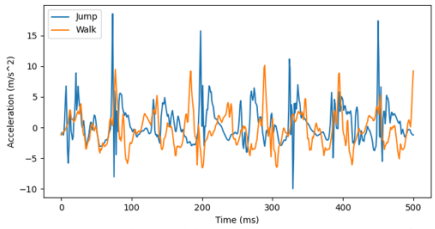


Figure 16: Real-world Pocket Acceleration for the Z Axis.

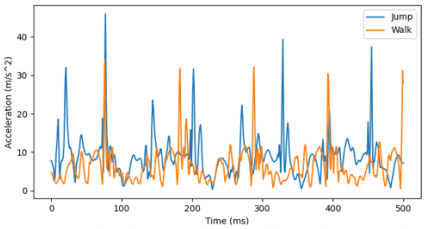


Figure 17: Absolute Real-world Pocket Acceleration.

Pre-processing

In almost every dataset, even when making extreme efforts to be careful during the data collection phase, there will still inevitably be noise. Data pre-processing is the concept of fixing minor or major issues with data, to ensure that sets are usable. This process could involve a variation of numerous approaches inclusive of imputation, normalization, feature extraction, dimensionality reduction, noise reduction, source separation, etc. [5].

In the case of this model a few of these pre-processing techniques were taken. The data collected contained plenty of noise. It is vital to minimize noise as it can destroy features in a signal. There are two categories, high-frequency noise, and low-frequency noise. In this case the noise obscuring the data is high frequency noise which adds fast moving noise to the signal. To reduce this type of noise a moving average (MA) filter is applied. MA is a low-pass filter, meaning it passes through low frequencies and discards high frequencies [5].

The idea of a MA filter is that they slide a window of a given size down the signal and average all the data points within that window. The filter will replace each value of the signal with the generated average value of values in a filter which precedes and includes said value [5]. The logic behind averaging the numbers is that the noise values cancel each other out.

The next approach was taken after the feature extraction. At this point an effort was made to detect and remove any outliers from the collected data. It was determined that, after analysing the de-noised data, none of the 5-second segments were far enough separate from each other to warrant treating those segments as outliers. As a result, the feature extraction and classification moved forward with the entire dataset. During the classification stage, trials were run with several subsets of the dataset to determine if any data was creating outliers, but the model's resulting accuracy was unaffected, reinforcing the lack of outliers in the normalized dataset.

Figure 18 below shows a graph of total acceleration vs time for a sample of walking data (blue) and a sample of jumping data (orange), Figure 19 shows the same sets of data graphed after a

moving average filter has been applied to the data. Viewing these graphs, it is clear that applying the moving average filter greatly reduced the noise in the data, removing many of the false peaks in acceleration, which results in a significantly more accurate model.

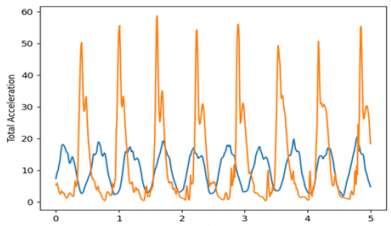


Figure 18: Absolute acceleration for a sample walking (blue) vs jumping (orange) data subset.

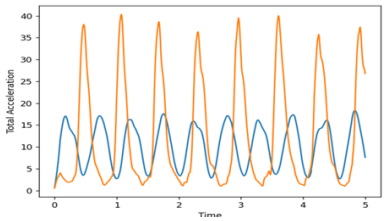


Figure 19: The same data as above, with a moving average applied to reduce noise.

Lastly, it was necessary to normalize the data so that it is fit for logistic regression. Code demonstrating these steps is displayed in the classifier program.

Feature Extraction

Feature extraction is the process of identifying and isolating various ‘features’ that contain key information. By utilizing these values, the whole signal is not required. During feature extraction there are a few design choices to be made, the first being window size. With a smaller window more information is extracted. The time window examined in this case is five second segments. Next there is window overlap. The model presented uses a rolling window. A rolling window moves ahead one sample at a time. This assists in retaining important information. Lastly and most importantly we must choose the types of features for extraction.

For this model, ten features were selected for extraction. The chosen features include, average x acceleration, maximum x acceleration, average y acceleration, maximum y acceleration, average z acceleration, maximum z acceleration, average time above the average z acceleration, overall average acceleration, overall maximum acceleration, and average time above the overall average acceleration. The goal in picking features is choosing a variety to summarize the important information contained in the original data sample [6].

Feature extraction addresses the issue of attaining the most informative and compact set of data we can, to improve the performance of a machine learning model. First, we consider ‘informative’. We are looking for features that can characterize the behavior of what we are trying to model [7]. In this case the end goal of the model is to classify a given data sample as either ‘walking’ or ‘jumping’. So, the strategy taken is to extract features that would be the most useful in defining a signal into one of those two categories.

What is meant by 'compact', is that we want to exclude irrelevant features from the model [7]. This proved to be an important consideration in the simple AI implementation. Throughout the design process several combinations of features were tested. It was discovered through trial and error that with too many feature extractions the model was less accurate in comparison to tests with fewer features. Further analysis would be necessary to select the distinct features of which were most beneficial in generating frequently successful results. Upon analyzing data visualizations of both jumping and walking samples and comparing the two, conclusive observations were made. It was determined that jumping data yielded acceleration vs. time graphs with overall higher and shorter peaks than those of walking samples. Therefore, a focus was set upon the average and maximum accelerations of each sample across all axes as well as averaged overall. This focus caused accuracy to improve drastically. A final ten features which yielded the highest accuracy were determined. Code implementing the feature extraction is shown below as Figure 20 to Figure 22. Note that the script shown below is extraction from the HDF5 file data.h5 from the group Ethan/Jump, the rest of this python file includes extraction from groups pertaining to every members data of both jumping and walking samples. Figure 23 represents the last few lines of code in the file writing to project_files.h5 in groups train and test.

```python
import h5py as h5
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split

empty = []
emptyset = np.array(empty)

#csv_file = "Feature_Extraction_Data.csv"
#info = ["avgXAccel", "maxXAccel",  "avgYAccel", "maxYAccel", "avgZAccel" ,"maxZAccel", "avgTimeAboveAvgZ", "avgAccel" , "maxAccel", "avgTimeAboveAvg", "label"]

#df.to_csv(csv_file, mode='a', header=False, index=False)


df = pd.DataFrame([emptyset])

with h5.File('./data.h5', 'r') as hdf:
    group = hdf.get('Ethan/Jump')
    group = list(group.keys())
for i in range (1, len(group)+1):
        with h5.File('./data.h5', 'r') as hdf:
            dataset = hdf.get('Ethan/Jump/' + str(group[i - 1]))
            myarray = np.array(dataset)
        #print(dataset)
        for i in range(0, len(myarray)):
            if i == 0:
                myarray[i][1] = myarray[i][1]
                myarray[i][2] = myarray[i][2]
                myarray[i][3] = myarray[i][3]
            else:
                myarray[i][1] = (myarray[i][1] + myarray[i - 1][1] + myarray[i - 2][1] + myarray[i - 3][1]) / 4
                myarray[i][2] = (myarray[i][2] + myarray[i - 1][2] + myarray[i - 2][2] + myarray[i - 3][2]) / 4
                myarray[i][3] = (myarray[i][3] + myarray[i - 1][3] + myarray[i - 2][3] + myarray[i - 3][3]) / 4

        # Average total acceleration
        avgAccel = 0
        for i in range(0, len(myarray)):
            avgAccel += abs(myarray[i][4] / 500)

        # max total acceleration
```

Figure 20: Part 1 of Python code from file Feature_Extraction_And_Storage .py which creates performs feature extraction and creates an HDF5 file called project_files.h5.

```
        maxAccel = 0
        for i in range(0, len(myarray)):
            if abs(myarray[i][4]) > maxAccel:
                maxAccel = abs(myarray[i][4])

        # Average X acceleration
        avgXAccel = 0
        for i in range(0, len(myarray)):
            avgXAccel += myarray[i][1] / 500

        # Average Y acceleration
        avgYAccel = 0
        for i in range(0, len(myarray)):
            avgYAccel += myarray[i][2] / 500

        # Average Z acceleration
        avgZAccel = 0
        for i in range(0, len(myarray)):
            avgZAccel += abs(myarray[i][3] / 500)
        # average time above avg Z acceleration

        avgZTime = 0
        timer = False
        times = 0
        totalTime = 0
        for i in range(0, len(myarray)):
            if abs(myarray[i][3]) >= avgZAccel:
                if timer == False:
                    timer = True
                    times += 1
                    totalTime += 1
                else:
                    totalTime += 1
            else:
                timer = False

        avgZTime = totalTime / times

        # max Z acceleration
        maxZAccel = 0
        for i in range(0, len(myarray)):
```

Figure 21: Part 2 of Python code from file Feature_Extraction_And_Storage .py which creates performs feature extraction and creates an HDF5 file called project_files.h5.

```
            if abs(myarray[i][3]) > maxZAccel:
                maxZAccel = abs(myarray[i][3])

        # max X acceleration
        maxXAccel = 0
        for i in range(0, len(myarray)):
            if abs(myarray[i][1]) > maxXAccel:
                maxXAccel = abs(myarray[i][1])

        # max Y acceleration
        maxYAccel = 0
        for i in range(0, len(myarray)):
            if abs(myarray[i][2]) > maxYAccel:
                maxYAccel = abs(myarray[i][2])

        # average time above avg acceleration

        avgTime = 0
        timer = False
        times = 0
        totalTime = 0
        for i in range(0, len(myarray)):
            if abs(myarray[i][4]) >= abs(avgAccel):
                if timer == False:
                    timer = True
                    times += 1
                    totalTime += 1
                else:
                    totalTime += 1
            else:
                timer = False

        avgTime = totalTime / times


        info = [avgXAccel, maxXAccel, avgYAccel, maxYAccel, avgZAccel, maxZAccel, avgZTime, avgAccel, maxAccel, avgTime, 1]
        df = pd.concat([df, pd.DataFrame([info])])
```

Figure 22: Part 3 of Python code from file Feature_Extraction_And_Storage .py which creates performs feature extraction and creates an HDF5 file called project_files.h5.

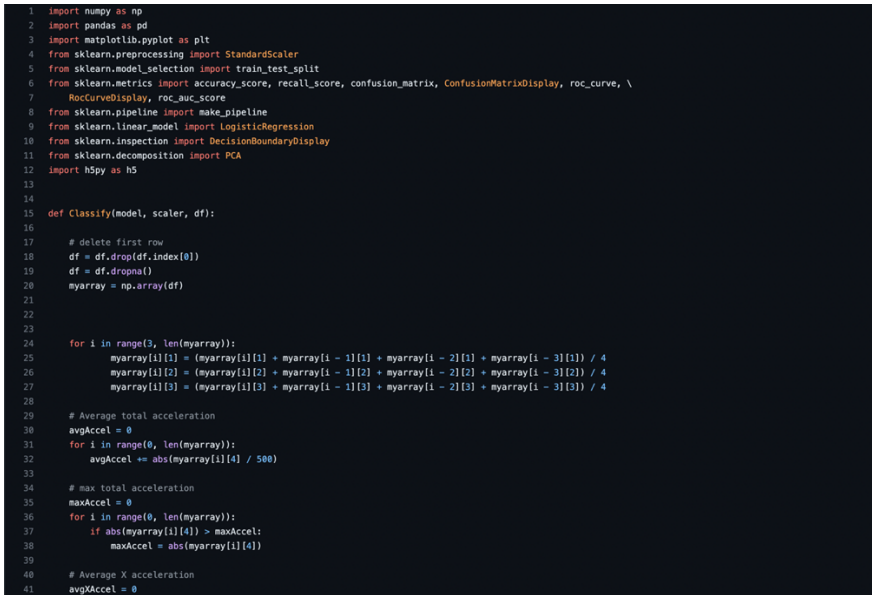
```
df.columns = ["avgXAccel", "maxXAccel",  "avgYAccel", "maxYAccel", "avgZAccel" ,"maxZAccel", "avgTimeAboveAvgZ", "avgAccel" , "maxAccel", "avgTimeAboveAvg", "label"]


Train, Test = train_test_split(df, test_size=0.1, shuffle=True)

with h5.File('./project_files.h5', 'w') as hdf:
    group = hdf.create_group('/dataset')
    group.create_dataset('train', data = Train)
    group.create_dataset('test', data = Test)
```

Figure 23: Part 4 of Python code from file Feature_Extraction_And_Storage .py which creates performs feature extraction and creates an HDF5 file called project_files.h5.

Training the Classifier

Using the features from the preprocessed training set, a logistic regression model was trained to classify the data into ‘walking’ and ‘jumping’ classes. During the training phase it was important to ensure that the test set didn’t leak into the training set. This implies that there was no overlap

between the segments used for training and those used for testing. Below are figures displaying the implementation of the classifier in python.

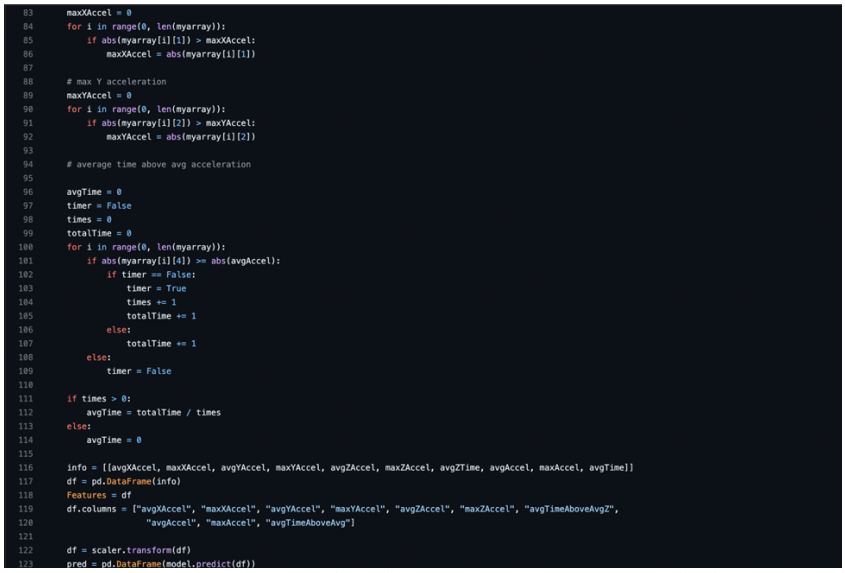


```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, confusion_matrix, ConfusionMatrixDisplay, roc_curve, \
    RocCurveDisplay, roc_auc_score
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.decomposition import PCA
import h5py as h5


def Classify(model, scaler, df):

    # delete first row
    df = df.drop(df.index[0])
    df = df.dropna()
    myarray = np.array(df)



    for i in range(3, len(myarray)):
            myarray[i][1] = (myarray[i][1] + myarray[i - 1][1] + myarray[i - 2][1] + myarray[i - 3][1]) / 4
            myarray[i][2] = (myarray[i][2] + myarray[i - 1][2] + myarray[i - 2][2] + myarray[i - 3][2]) / 4
            myarray[i][3] = (myarray[i][3] + myarray[i - 1][3] + myarray[i - 2][3] + myarray[i - 3][3]) / 4

    # Average total acceleration
    avgAccel = 0
    for i in range(0, len(myarray)):
        avgAccel += abs(myarray[i][4] / 500)

    # max total acceleration
    maxAccel = 0
    for i in range(0, len(myarray)):
        if abs(myarray[i][4]) > maxAccel:
            maxAccel = abs(myarray[i][4])

    # Average X acceleration
    avgXAccel = 0
```

Figure 24: Part 1 of Python code from file CLASSIFIER_FINAL .py which creates performs training on the classifier.

```python
    for i in range(0, len(myarray)):
        avgXAccel += myarray[i][1] / 500

    # Average Y acceleration
    avgYAccel = 0
    for i in range(0, len(myarray)):
        avgYAccel += myarray[i][2] / 500

    # Average Z acceleration
    avgZAccel = 0
    for i in range(0, len(myarray)):
        avgZAccel += abs(myarray[i][3] / 500)
    # average time above avg Z acceleration

    avgZTime = 0
    timer = False
    times = 0
    totalTime = 0
    for i in range(0, len(myarray)):
        if abs(myarray[i][3]) >= abs(avgZAccel):
            if timer == False:
                timer = True
                times += 1
                totalTime += 1
            else:
                totalTime += 1
        else:
            timer = False

    if times > 0:
        avgZTime = totalTime / times
    else:
        avgZTime = 0

    # max Z acceleration
    maxZAccel = 0
    for i in range(0, len(myarray)):
        if abs(myarray[i][3]) > maxZAccel:
            maxZAccel = abs(myarray[i][3])

    # max X acceleration
```

Figure 25: Part 2 of Python code from file CLASSIFIER_FINAL .py which creates performs training on the classifier.



```
    maxXAccel = 0
    for i in range(0, len(myarray)):
        if abs(myarray[i][1]) > maxXAccel:
            maxXAccel = abs(myarray[i][1])

    # max Y acceleration
    maxYAccel = 0
    for i in range(0, len(myarray)):
        if abs(myarray[i][2]) > maxYAccel:
            maxYAccel = abs(myarray[i][2])

    # average time above avg acceleration

    avgTime = 0
    timer = False
    times = 0
    totalTime = 0
    for i in range(0, len(myarray)):
        if abs(myarray[i][4]) >= abs(avgAccel):
            if timer == False:
                timer = True
                times += 1
                totalTime += 1
            else:
                totalTime += 1
        else:
            timer = False

    if times > 0:
        avgTime = totalTime / times
    else:
        avgTime = 0

    info = [[avgXAccel, maxXAccel, avgYAccel, maxYAccel, avgZAccel, maxZAccel, avgZTime, avgAccel, maxAccel, avgTime]]
    df = pd.DataFrame(info)
    Features = df
    df.columns = ["avgXAccel", "maxXAccel", "avgYAccel", "maxYAccel", "avgZAccel", "maxZAccel", "avgTimeAboveAvgZ",
                  "avgAccel", "maxAccel", "avgTimeAboveAvg"]

    df = scaler.transform(df)
    pred = pd.DataFrame(model.predict(df))
```

Figure 26: Part 3 of Python code from file CLASSIFIER_FINAL .py which creates performs training on the classifier.

```
    final = pd.concat([Features, pred], axis=1)
    final.columns = ["avgXAccel", "maxXAccel", "avgYAccel", "maxYAccel", "avgZAccel", "maxZAccel", "avgTimeAboveAvgZ",
                     "avgAccel", "maxAccel", "avgTimeAboveAvg", 'label']
    return final




def train():
    with h5.File('./project_files.h5', 'r') as hdf:
        Train = pd.DataFrame(hdf.get('/dataset/train'))
        Test = pd.DataFrame(hdf.get('/dataset/test'))

    Train = Train.dropna()
    Test = Test.dropna()

    Train.columns = ["avgXAccel", "maxXAccel", "avgYAccel", "maxYAccel", "avgZAccel", "maxZAccel", "avgTimeAboveAvgZ",
                     "avgAccel", "maxAccel", "avgTimeAboveAvg", "label"]
    Test.columns = ["avgXAccel", "maxXAccel", "avgYAccel", "maxYAccel", "avgZAccel", "maxZAccel", "avgTimeAboveAvgZ",
                    "avgAccel", "maxAccel", "avgTimeAboveAvg", "label"]

    Y_train = Train['label']
    Y_test = Test['label']

    X_train = Train.iloc[:, :-1]
    X_test = Test.iloc[:, :-1]

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

    # Now, similar to the code presented in the lecture slides, train and test your model.

    l_reg = LogisticRegression(max_iter=10000)
    clf = make_pipeline(StandardScaler(), l_reg)

    clf.fit(X_train, Y_train)

    return clf, scaler
```

Figure 27: Part 4 of Python code from file CLASSIFIER_FINAL .py which creates performs training on the classifier.

The classifier shown in the figures above includes two main functions, classify and train. Classify performs some preprocessing and feature extraction on the input df, which contains accelerometer data. Any rows with missing or irrelevant values are dropped. After calculating the features based on given accelerometer readings, they are then combined into a new DataFrame. The new DataFrame ‘Features’ is transformed using ‘scaler’ and passed to ‘model’ for prediction. The train function reads in project_files.h5 which was created in the previous step and trains a logistic regression model using make_pipeline.

Moreover, a Python script CLASSIFIER_VISUALS.py was written for training and testing. The accelerometer data from project_files.h5. is read and loaded into data frames ‘Train’ and ‘Test’. Accelerometer data is separated and stored in X_train and X_test, which are normalized using

StandardScaler. The logistic regression model is set with a maximum iteration of 1000 and the pipeline is fitted to the training data. In this program the accuracy and recall scores are calculated and printed which is shown in Figure 28. A ROC curve is created using the predicted probabilities and the actual labels from test data. The area under the ROC curve, the AUC is calculated and printed which is also visible in Figure 28. Finally, a confusion matrix is developed as well, which is shown in Figure 30.

```
y_pred is:  [0. 1. 0. 0. 1. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 0. 1. 1. 1.]
y_clf_prob is:  [[9.07512004e-01 9.24879961e-02]
 [1.60073895e-03 9.98399261e-01]
 [9.91423473e-01 8.57652726e-03]
 [9.73944546e-01 2.60554542e-02]
 [2.82729533e-02 9.71727047e-01]
 [8.89742738e-01 1.10257262e-01]
 [1.81965505e-04 9.99818034e-01]
 [9.05957529e-04 9.99094042e-01]
 [7.52396148e-01 2.47603852e-01]
 [5.70366207e-01 4.29633793e-01]
 [9.91391093e-01 8.60890652e-03]
 [8.96903208e-01 1.03096792e-01]
 [3.61818873e-03 9.96381811e-01]
 [3.25641972e-02 9.67435803e-01]
 [9.84493277e-01 1.55067230e-02]
 [9.89166862e-01 1.08331375e-02]
 [3.35967857e-03 9.96640321e-01]
 [4.91731529e-01 5.08268471e-01]
 [2.14853742e-03 9.97851463e-01]]
Accuracy is:  0.9473684210526315
0.9
AUC is:  1.0
```

Figure 28: Output of CLASSIFIER_VISUALS.py displaying accuracy and AUC.

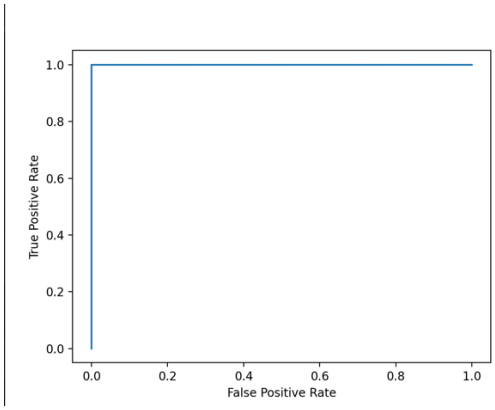


Figure 29: ROC developed from CLASSIFIER_VISUALS.py.

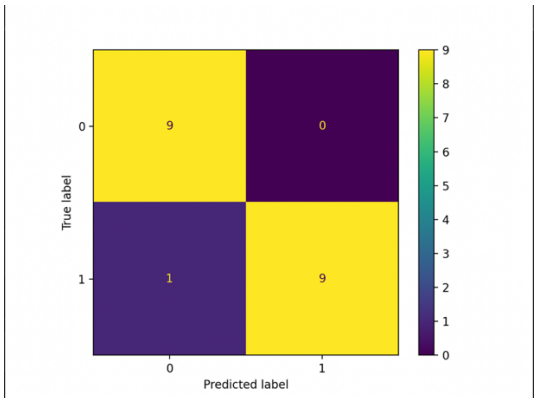


Figure 30: Confusion matrix developed by CLASSIFIER_VISUALS.py.

The code to accomplish these data and figures is displayed in Figure 31 and Figure 32.

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, recall_score, confusion_matrix, ConfusionMatrixDisplay, roc_curve, \
    RocCurveDisplay, roc_auc_score
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.decomposition import PCA
import h5py as h5
import numpy as np
import pandas as pd

# Step 2. Read the dataset as a Pandas data frame
with h5.File('./project_files.h5', 'r') as hdf:
    Train = pd.DataFrame(hdf.get('/dataset/train'))
    Test = pd.DataFrame(hdf.get('/dataset/test'))

Train = Train.dropna()
Test = Test.dropna()

Train.columns = ["avgXAccel", "maxXAccel",  "avgYAccel", "maxYAccel", "avgZAccel", "maxZAccel", "avgTimeAboveAvgZ", "avgAccel", "maxAccel", "avgTimeAboveAvg", "la
Test.columns = ["avgXAccel", "maxXAccel",  "avgYAccel", "maxYAccel", "avgZAccel", "maxZAccel", "avgTimeAboveAvgZ", "avgAccel", "maxAccel", "avgTimeAboveAvg", "lab

Y_train = Train['label']
Y_test = Test['label']

X_train = Train.iloc[:, :-1]
X_test = Test.iloc[:, :-1]


scaler = StandardScaler()
```

Figure 31: Python file CLASSIFIER_VISUALS.py part 1.

```python
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Now, similar to the code presented in the lecture slides, train and test your model.

l_reg = LogisticRegression(max_iter=10000)
clf = make_pipeline(StandardScaler(), l_reg)

clf.fit(X_train, Y_train)

y_pred = clf.predict(X_test)
y_clef_prob = clf.predict_proba(X_test)
print("y_pred is: ", y_pred)
print("y_clf_prob is: ", y_clef_prob)

acc = accuracy_score(Y_test, y_pred)
print("Accuracy is: ", acc)
recall = recall_score(Y_test, y_pred)
print(recall)

cm = confusion_matrix(Y_test, y_pred)
cm_display = ConfusionMatrixDisplay(cm).plot()
plt.show()

fpr, tpr, _ = roc_curve(Y_test, y_clef_prob[:, 1], pos_label=clf.classes_[1])
roc_display = RocCurveDisplay(fpr=fpr, tpr=tpr).plot()
plt.show()

auc = roc_auc_score(Y_test, y_clef_prob[:, 1])
print("AUC is: ", auc)
```

Figure 32: Python file CLASSIFIER_VISUALS.py part 2.

Model Deployment

The final step of the design process was to deploy the model in a desktop app. The goal was to have the application accept an input file in CSV format and generate a CSV file as the output, which would include the prediction label, either walking or jumping, for the selected input file. To build the graphical user interface the Python library Tkinter was used. The overall design is straight forward. The interface provides a select file button to choose a CSV file which would contain accelerometer data. The GUI also has buttons to then generate output based on data provided in the selected CSV. The Application's initializer method sets up the various buttons, frames, as well as a graph. It also loads the trained model from CLASSIFIER_FINAL.py. When the 'Select CSV File' button is clicked the user will see a file dialog box. When the file is chosen, it gets read to a DataFrame, and split into segments of 500 rows each. Both buttons to generate output are enabled when needed. The main window is shown in Figure 33.

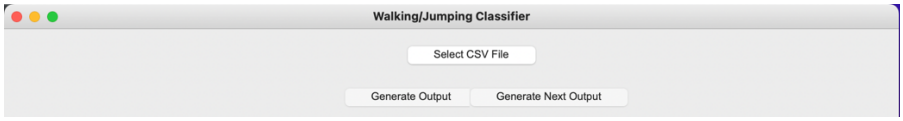


Figure 33: Main window of simple GUI.

Upon generating the output, the prediction label and output graph are displayed. The prediction label will simply tell a user if the file they selected contained walking or jumping data. The figure generated shows a bar graph with each of the extracted features. An example of the generated output is shown below in Figure when a file called 'jumping_backpocketL1.csv' is uploaded to the interface. Though it should be noted that several tests were conducted with a wide variety of input files uploaded to ensure functionality. Another measure taken was to test walking and jumping data from individuals outside of the team. Meaning the model was successful for people who may have a different stride, higher jump etc. than the team members who built the model with their walking/jumping data.

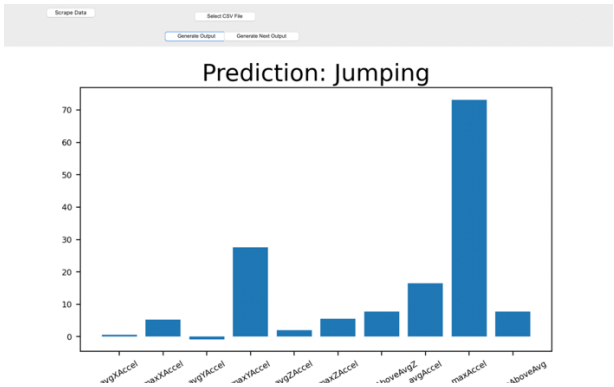


Figure 34: Main window after pressing the 'Generate Output' button with a file uploaded called 'jumping_backpocketL1.csv'.

Real Time

After development of the model deployment, a real time implementation was attempted. The goal of this change would be to create an application which is capable of reading accelerometer data from a smart phone and classifying it in real time. To implement this a different button was created on the main window which allowed a user to scrape data from a web page.

```
import tkinter as tk
from tkinter import ttk as ttk
import pandas as pd
import matplotlib.pyplot as plt
from tkinter.filedialog import askopenfilename
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import classifier
import numpy as np
import math
import time
from bs4 import BeautifulSoup
from selenium import webdriver

class Application:

    def __init__(self, master):
        self.master = master
        self.master.title("Walking/Jumping Classifier")


        # scrape button
        self.scrape_button = ttk.Button(master, text="Scrape Data", command=self.scrape_data)
        self.scrape_button.place(x=100, y=5)
```

Figure 35: Initialization of necessary libraries and a new button on the main window.

The approach taken was to enable remote access through the Phyphox app which is used for the data collection process. The smart phone on which data is being collected can then connect to a web page correlating to the IP address of that device this is seen in line 70 of Figure 36 as ‘http://192.168.0.212’. In order to scrape information from the Phyphox page the Python libraries Selenium and Beautiful Soup are included. Upon clicking the scrape button, a method is called initialize a Selenium web driver retrieving the Phyphox page. The Beautiful Soup library is used to try colleting the necessary data which can be put into a data frame. A while loop continuously extracts the latest data from the page. Lastly, using the previously established classifier a prediction is returned to tell a user which motion is being demonstrated. With further time and iteration, the real time implementation could have been fully developed.

```
66      def scrape_data(self):
67          # Web driver
68          driver = webdriver.Chrome()
69          # link correlating to the IP of the smart-phone
70          driver.get("https://192.168.0.212")
71          # load delay
72          time.sleep(1)
73
74          # Finding the necessary data
75          soup = BeautifulSoup(driver.page_source, "html.parser")
76          table = soup.find("table", {"id": "data-table"})
77          data = []
78          for row in table.find_all("tr"):
79              cols = row.find_all("td")
80              cols = [col.text.strip() for col in cols]
81              data.append(cols)
82
83          while True:
84              # Extract the latest data
85              html = driver.page_source
86              soup = BeautifulSoup(html, "html.parser")
87              table = soup.find("table", {"id": "data-table"})
88              last_data = table.find_all("tr")[-1]
89              last_values = [float(col.text.strip()) for col in last_data.find_all("td")[1:]]
90
91              # Create a DataFrame df with the most recent data
92              df = pd.DataFrame([last_values], columns=["Time", "X-Accel", "Y-Accel","Z-Accel"])
93              df = df.rolling(window=4).mean().dropna()
```

Figure 36: Part 1 of Python code implemented to attempt real time classification.

```
94
95              # Delay
96              time.sleep(0.1)
97              # Classify data
98              pred = classifier.Classify(self.model, self.scaler, df)
99
100             # Print the predicted motion
101             if pred[0] == 1:
102                 print("Walking")
103             else:
104                 print("Jumping")
105
106             print(pred)
```

Figure 37: Part 2 of Python code implemented to attempt real time classification.

Participation Report

A summary of group participation can be seen below in Table 1.

Table 1: Participation table summarizing contributions.

Program	
Section	Member(s)
Data Collection	Ethan, Lauren, Jacob
Data Storage	Ethan, Lauren, Jacob
Visualization	Ethan
Pre-processing	Ethan, Jacob
Feature Extraction	Ethan, Jacob
Creating a Classifier	Ethan, Jacob
Deployment	Lauren, Jacob
Real Time	Lauren
Report	
Section	Member(s)
Data Collection	Lauren
Data Storage	Lauren, Ethan
Visualization	Lauren, Ethan
Pre-processing	Lauren, Jacob, Ethan
Feature Extraction	Lauren, Jacob
Training the Classifier	Lauren, Jacob
Deployment	Lauren
Real Time	Lauren
Video	
Section	Member(s)
Recording	Ethan, Jacob, Lauren
Editing	Ethan

Bibliography

[1] A. Etemad, "ELEC 390 Principles of Design & Dev. W23 Slides - Part 1," 2023. [Online]. Available: https://onq.queensu.ca/d2l/le/content/711190/viewContent/4584784/View. [Accessed 7 April 2023].

[2] A. Etemad, "ELEC 390 Principles of Design & Dev. W23 Slides - Part 2," 2023. [Online]. Available: https://onq.queensu.ca/d2l/le/content/711190/viewContent/4618175/View. [Accessed 7 April 2023].

[3] A. Etemad, "ELEC 390 Principles of Design & Dev. W23 Project Instructions," 2023. [Online]. Available: https://onq.queensu.ca/d2l/le/content/711190/viewContent/4586805/View. [Accessed 24 Feb 2023].

[4] A. Etemad, "ELEC 390 Principles of Design & Dev. W23 Slides - Part 3," 2023. [Online]. Available: https://onq.queensu.ca/d2l/le/content/711190/viewContent/4634633/View. [Accessed 7 April 2023].

[5] A. Etemad, "ELEC 390 Principles of Design & Dev. W23 Slides - Part 4," 2023. [Online]. Available: https://onq.queensu.ca/d2l/le/content/711190/viewContent/4647961/View. [Accessed 7 April 2023].

[6] p. Ippolito, "Feature Extraction Techniques," Medium, 10 October 2019. [Online]. Available: https://towardsdatascience.com/feature-extraction-techniques-d619b56e31be. [Accessed 12 April 2023].

[7] P. Marcelino, "Data Analysis and Feature Extraction with Python," Kaggle, 2022. [Online]. Available: https://www.kaggle.com/code/pmarcelino/data-analysis-and-feature-extraction-with-python. [Accessed 12 April 2023].