Final Project: Walking/Jumping Classification

ELEC 292

Faculty of Engineering and Applied Science

Queen’s University

**Prepared by [Group 58]**

Anapayaan Pakerathan – 20347629 – 21arp10@queensu.ca

Hendrix Gryspeerdt – 20337154 – 21hgg3@queensu.ca

Sunday April 7th, 2024

*“We do hereby verify that this written report is our own individual work and contains our own original ideas. No portion of this report has been copied in whole or in part from another source,* *with the possible exception of properly referenced material.*

# 1. Data Collection

The data collection process involved working with an app called phyphox denoted as an application mainly for phones with various sensors. These sensors serve to collect input from users and provide an in-depth breakdown of their movements. With several graphs depicting time measurements, motion analysis, sound intensity and numerous other data visualizations. Phyphox offers a platform to conduct these experiments and explore the physical aspects of various phenomena with accuracy and simplicity. Data was collected from the raw sensors specifically acceleration(without a gravitational force) provided by the linear accelerometer. The data included measurements of linear acceleration across all dimensions in the x-y-z plane respectively as well as absolute acceleration with time being the independent variable. The user input involved all members of the team to record their own data inorder to eliminate any bias and effectively compare results, ensuring the credibility and reliability of the findings. Each individual was required to walk and jump in 40 second intervals in various positions ranging from front to back, chest to hand pocket. The total duration of the data collection process was 5 minutes and 20 seconds. Upon collection of the data an export feature in the phyphox application was done to save the measurements as a single csv file alongside the uploaded metadata. The metadata essentially consisted of information about the device with specific references to the time the data had been performed. It’s important to note that during the data collection process files were exported in 40 second intervals, resulting in a total of 8 csv files for each individual.

A screenshot of a black screen

Description automatically generated

Figure 1: Data collection protocol.

# 2. Data Storing

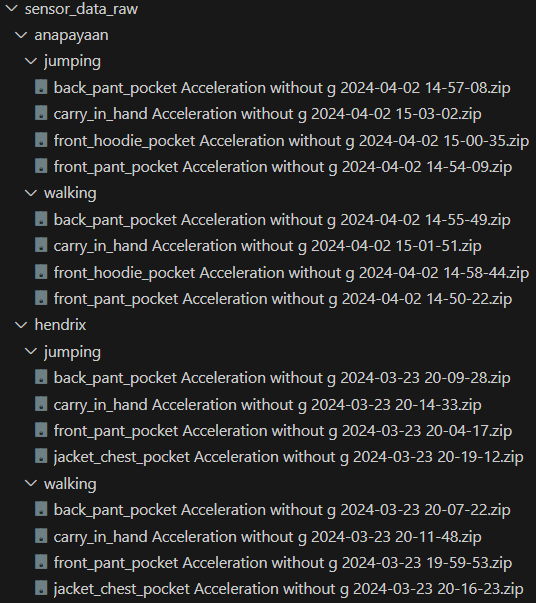
Data storing occurred in 3 stages. The first stage of data was to organize the direct email exports from phyphox stored in zip files. The second stage of data resulted from unzipping the compressed files from phyphox. The third stage was to store all datasets in an hdf5 file in the format as collected and in a format more suited for training and testing a classification model. A diagram illustrating this process is shown in Figure 2 below.

A diagram of a computer

Description automatically generated

Figure 2: Stages in the data storage process.

The resulting file structure of the first two stages of data storage are shown in the two figures below.

 A screenshot of a computer

Description automatically generated

Figure 3: (left) zip files as exported from phyphox [1]. (right) File structure of compressed data exported from phyphox [1].

The acceleration data in each time-stamped directory is found in the file “Raw Data.csv”. Additional metadata regarding the device used to collect data is found under the meta subdirectory in files “device.csv” and “time.csv”. The headers of those files are shown below in Figure 4 .

|  |  |
| --- | --- |
| A screenshot of a computer  Description automatically generated | A screenshot of a computer  Description automatically generated |

Figure 4: Files located in directory "sensor\_data/csv/hendrix/jumping/back\_pant\_pocket Acceleration without g 2024-03-23 20-09-28". “meta/device.csv” on the left, “meta/time.csv” on the top, and “Raw Data.csv” is on the bottom. The larger figure is truncated as there are thousands of rows.

Figure 5: metadata for anapayaan's smartphone accelerometer. File is large so it is truncated in this figure. File name is "sensor\_data\csv\anapayaan\jumping\back\_pant\_pocket Acceleration without g 2024-04-02 14-57-08\meta\device.csv".

Metadata indicates that sampling rates for the different devices may differ. This is accounted for by ensuring windows sizes are based on the “Time (s)” column and not just the number of rows.

The file data\_storing.py contains the functions used to process the data for storage. The function *unzip\_sensor\_data()* was used to extract the zip files. The second function, *create\_hdf5()* is used to then walk through the “sensor\_data/csv” directory to create an HDF5 file with the structure shown in Figure 6.

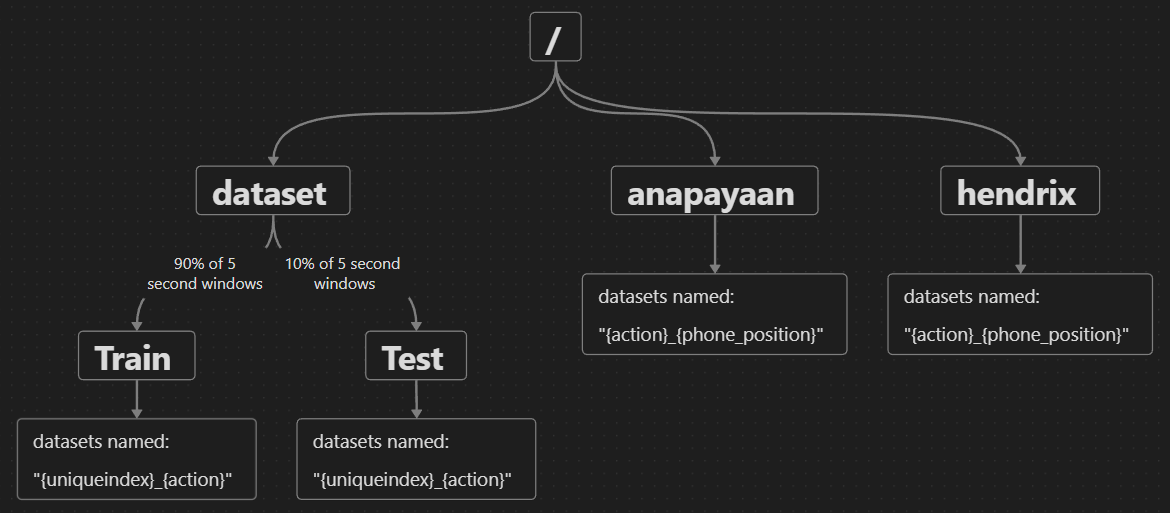


Figure 6: Structure of data stored in file “sensor\_data.hdf5”. All datasets have the same header as the bottom image in Figure 4.

Each dataset from files named “Raw Data.csv” are stored in the HDF5 file using the following code snippet from create\_hdf5().

A screen shot of a computer program

Description automatically generated

Figure 7: create\_hdf5() function located in data\_storing.py.

Slicing the datasets using 5 second windows significantly increase the amount of data being stored due to overlap of the windows. This results in “sensor\_data.hdf5” having a large file size of near 3.7 GB. This is significantly more than the data 9.67MB of data stored in raw csv format in the entire directory “sensor\_data/csv”.

The last two functions in data\_storage.py are used to load the Train and Test datasets into Pandas Dataframe Objects [2] for the purposes of training and testing the model. Loading the datasets into Pandas Dataframes the sizes of datasets can be identified. The code snippet in Figure 8 shows the different shapes of the structures.

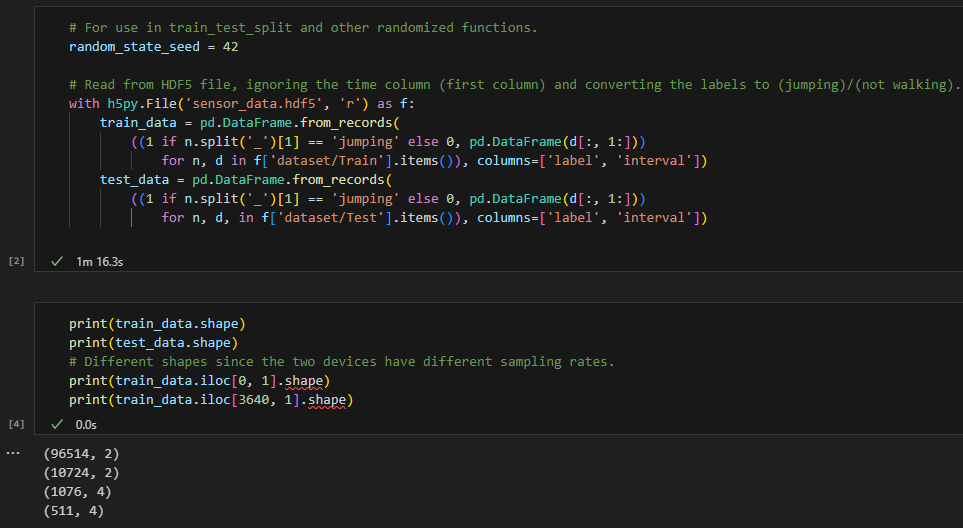


Figure 8: Loading data to show the different interval sizes as a result of accelerometer sampling rates.

This concludes data storing, data is now organized and ready to be analysed and used to train a binary classification model. Since the HDF5 file is too large to be uploaded on OnQ, you need to run “data\_storing.py” to load the hdf5 file on the resulting computer.

# 3. Visualization

A screen shot of a computer screen

Description automatically generated

*Figure 9 Walking with phone in hand (Anapayaan)*

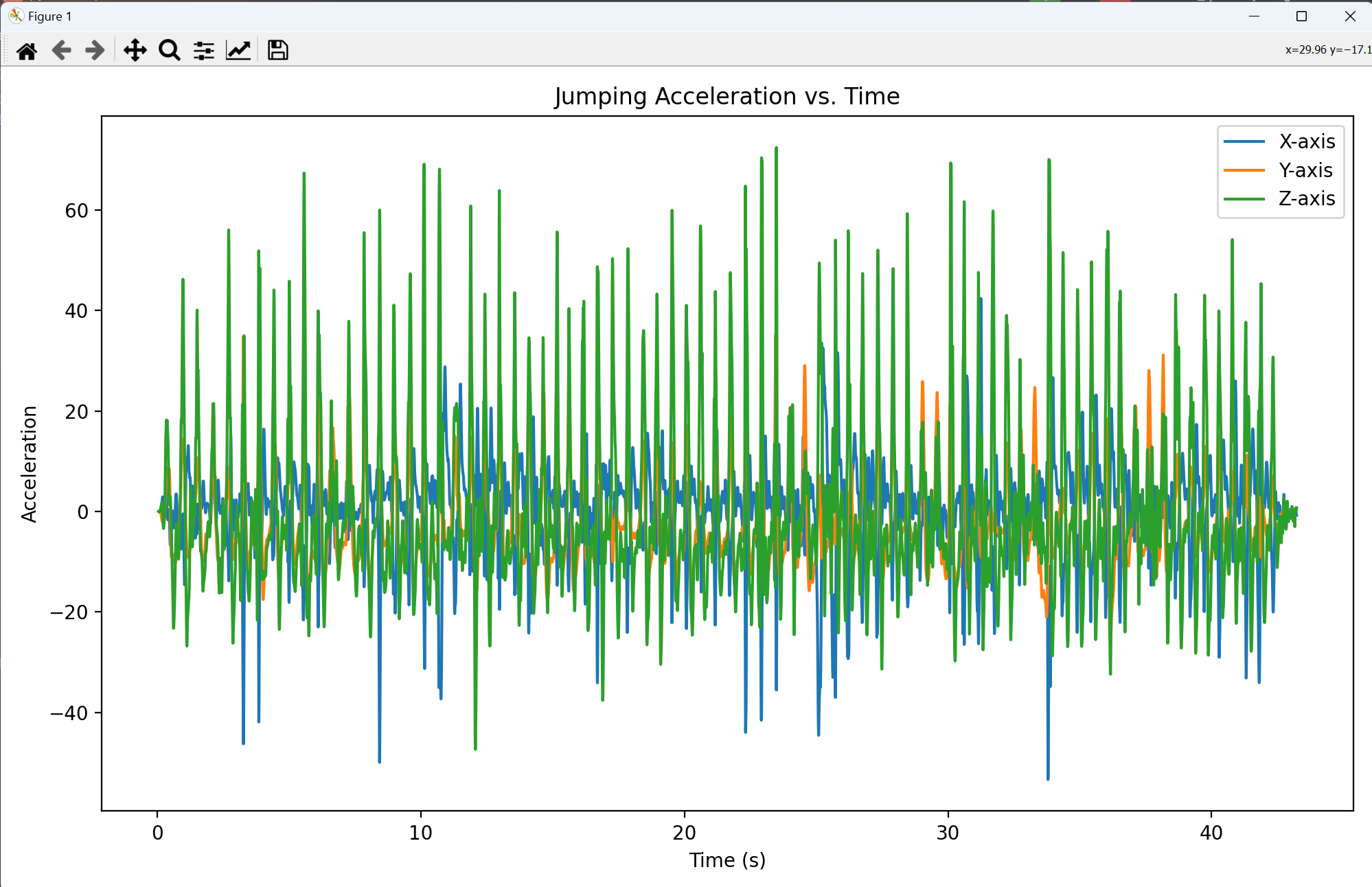


Figure 10 Jumping with phone in hand (Anapayaan)

A screen shot of a computer screen

Description automatically generated

Figure 11 Walking with phone in front pocket (Anapayaan)

A screen shot of a graph

Description automatically generated

Figure 12 Jumping with phone in front pocket (Anapayaan)

A screen shot of a computer screen

Description automatically generated

Figure 13 Walking with phone in hand (Hendrix)

A screen shot of a graph

Description automatically generated

Figure 14 Jumping with phone in hand (Hendrix)

A screen shot of a computer screen

Description automatically generated

Figure 15 Walking with phone in front pocket (Hendrix)

A screen shot of a computer screen

Description automatically generated

Figure 16 Jumping with phone in front pocket (Hendrix)

A graph showing a bunch of tangled strings

Description automatically generated

Figure 17 3D Trajectory Plot of Walking with phone in hand

**Data Visualization Code:**

import tkinter as tk  
from tkinter import filedialog  
import pandas as pd  
import matplotlib.pyplot as plt  
from mpl\_toolkits.mplot3d import Axes3D  
  
def upload\_csv():  
 file\_path = filedialog.askopenfilename(filetypes=[("CSV files", "\*.csv")])  
 if file\_path:  
 # Load dataset  
 data = pd.read\_csv(file\_path)  
  
  
  
 plt.figure(figsize=(10, 6))  
  
 # Walking Acceleration vs. Time  
 plt.subplot(1, 1, 1)  
 plt.plot(data['Time (s)'], data['Linear Acceleration x (m/s^2)'], label='X-axis')  
 plt.plot(data['Time (s)'], data['Linear Acceleration y (m/s^2)'], label='Y-axis')  
 plt.plot(data['Time (s)'], data['Linear Acceleration z (m/s^2)'], label='Z-axis')  
 plt.title('Walking Acceleration vs. Time')  
 plt.xlabel('Time (s)')  
 plt.ylabel('Acceleration')  
 plt.legend()  
 plt.tight\_layout()  
 plt.show()  
  
 # Additional Creative Visualization Ideas  
 # 3D Trajectory Plot  
 fig = plt.figure()  
 ax = fig.add\_subplot(111, projection='3d')  
 ax.plot(data['Linear Acceleration x (m/s^2)'], data['Linear Acceleration y (m/s^2)'], data['Linear Acceleration z (m/s^2)'],  
 label='Walking')  
 ax.plot(data['Linear Acceleration x (m/s^2)'], data['Linear Acceleration y (m/s^2)'], data['Linear Acceleration z (m/s^2)'],  
 label='Jumping')  
 ax.set\_xlabel('X-axis Acceleration')  
 ax.set\_ylabel('Y-axis Acceleration')  
 ax.set\_zlabel('Z-axis Acceleration')  
 ax.set\_title('3D Trajectory Plot')  
 plt.show()  
  
 # Load metadata (assuming metadata.csv is in the same directory as the uploaded CSV file)  
 metadata\_path = file\_path.replace('.csv', '\_metadata.csv')  
 metadata = pd.read\_csv(metadata\_path)  
  
 # Visualization of Meta-Data  
 # Histogram of Sampling Rates  
 plt.hist(metadata['sampling\_rate'], bins=10)  
 plt.title('Distribution of Sampling Rates')  
 plt.xlabel('Sampling Rate')  
 plt.ylabel('Frequency')  
 plt.show()  
  
 # Box Plots for Sensor Locations  
 plt.boxplot([metadata['sensor\_x'], metadata['sensor\_y'], metadata['sensor\_z']], labels=['X', 'Y', 'Z'])  
 plt.title('Sensor Locations')  
 plt.xlabel('Axis')  
 plt.ylabel('Position')  
 plt.show()  
  
# Create the main window  
root = tk.Tk()  
root.title("CSV File Upload")  
  
# Create a button to upload CSV file  
upload\_button = tk.Button(root, text="Upload CSV File", command=upload\_csv)  
upload\_button.pack(pady=20)  
  
# Run the Tkinter event loop  
root.mainloop()

In terms of data visualization there are numerous things I learned about this process from the significance of data pre-processing to the specific selection of various graphs to be used can ultimately hinder the quality of visualizations. With multiple methods for expressing data, it’s crucial to really understand what information is most important and how the reflection of various graphs outline those features. It’s honestly a big learning curve and requires a lot of strategizing and planning to figure out the most optima way to display the significance of information. If data collection were to be performed again there would be more attention towards the general audience and how the data outputted can be more easily interpretable. There would also exist more design features like aesthetics and layout contributing to the overall user experience.

# 4. Pre-processing

- just doing moving average to smooth out any potential noise from the accelerometers

The purpose of Pre-processing was to reduce noise produced by the accelerometers. Ideally, pre-processing would also include outlier removal, however, this was not done due to limited time and since the classifier was already quite accurate (as will be seen in section 6. Creating a Classifier).

As covered in class, the moving average filter is effective in reducing high-frequency noise in the dataset. However, since the accelerometer data is already high-frequency, too-large a window size would eliminate too much information. Therefore, a small window size of 10 was selected for applying the simple moving average. Examples of applying this moving average filter on 5 second interval of jumping and walking data are shown in the following table.

|  |  |  |
| --- | --- | --- |
|  | Before Pre-Processing | After Pre-Processing |
| Jumping |  |  |
| Walking |  |  |

Figure 18: Illustrating the effects of the moving average filter on the accelerometer data. Graphs produce in sections 3 and 4 in the jupyter notebook classifier.ipynb.

Other forms of outlier removal could have been done, such as excluding the first and last 3 seconds of each csv file from being used. This would eliminate the time the spent to take the phone in/out of the pocket when starting/ending data collection. Overall, the sensor data was quite consistent, this led to little pre-processing being necessary. Final implementation of pre-processing can be found in the file classifier.py.

# 5. Feature Extraction & Normalization

After the data was pre-processed, each 5 second interval had to have features extracted to reduce the dimensionality of the 500-1000 timesteps and the 4 acceleration columns in x, y, z coordinates and absolute acceleration to something more manageable for a logistic regression model. The choice then is to calculate a short list of features for each 5 second window of acceleration data.

The features extracted for each column were min, max, range, mean, median, standard deviation, and variance, for a total of 7 different features per column. The feature extraction functions were supplied by the Pandas library. With 4 columns and 7 features per column, this resulted in a total of 28 features per 5 second interval.

After extracting the features, any intervals that resulted in missing values (NaN) were removed. Normalizing the features was then considered so to not have certain features of higher magnitude disproportionately influence the results of classification. Min-Max Scaling was applied to ensure all features were in the range from zero to one. Z-Score standardization did not seem reasonable here since the features were not expected to form a normal distribution and this normalization technique does not ensure that all the features have values in a consistent range. Hence the MinMaxScalar [3] normalization method was used as was supplied by the scikit-learn [4] python library.

Normalization was not done on its own, as it made sense to combine normalization and logistic regression into one pipeline for the classifier. Final implementation of normalization can be found in the classifier\_create() function of classifier.py and whereas experimentation can be found in section 6. Creating a Classifier in classifier.ipynb.

# 6. Creating a Classifier

The problem of classifying between two possible actions, walking and jumping, is a binary classification problem, and as taught in class, logistic regression models are effective at performing this task.

After the features were extracted they were fed into the classification model pipeline which included MinMaxScaler() and LogisticRegression() whose implementations were provided by the scikit-learn library.

Once the model was fit to the data, the classifier model was saved to a binary file classifier.pkl using the python pickle library pickle.dump [5] function. This allowed for future re-use without the need for re-training.

The creation and training of the model was completed and all implementation details can be found in the function classifier\_create() in classifier.py file.

To test the classifier to see how it performed on the test split of the data. The classifier\_test() function is used. The testing data is loaded from the hdf5 file “sensor\_data.hdf5”; the model is deserialized from the save file “classifier.pkl” using pickle.load [5]. Then predictions are performed to calculate Accuracy, Recall, F1 Score, and Confusion Matrix, as well as ROC AUC and ROC Curves were calculated. Outputs from running classifier\_test() can be seen in the figures below.

A screenshot of a computer screen

Description automatically generated

Figure 19: Console output from classifier\_test(), listing model evaluation metrics.

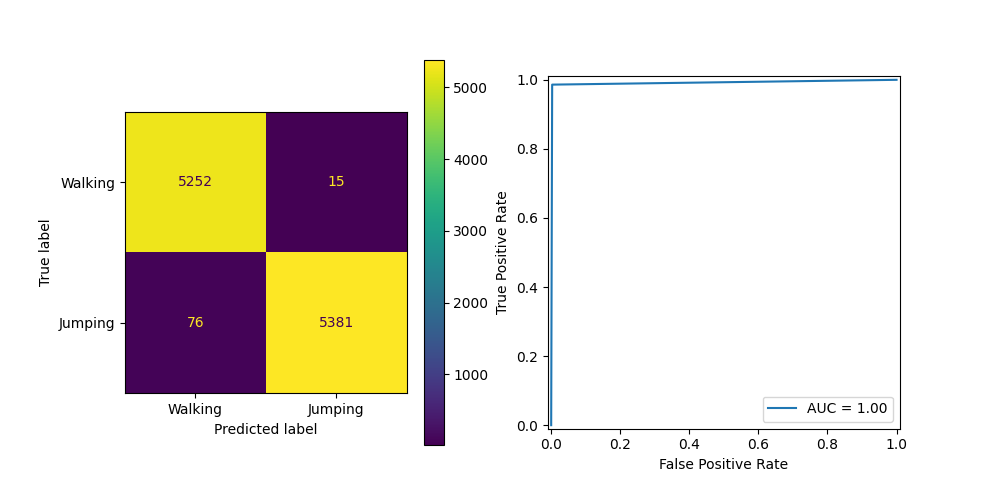


Figure 20: (left) Confusion matrix, (right) ROC curve. These figures show how the model performed.

Below is the code which was used to achieve the results in training and testing. Separation of training and testing data was ensured by only loading the training data when training occurred and only loading testing data when testing.

A screen shot of a computer program

Description automatically generated

Figure 21: Python code for classifier\_create() and associated functions in the classifier.py file.

A screen shot of a computer program

Description automatically generated

Figure 22: classifer\_test() function as found in the classifier.py file.

This concludes training and testing the model. From the results were successful due to the high scores for the various evaluation metrics mentioned earlier in this section.

# 7. Model Deployment

Before the model can be deployed and the app can be run, you must have run the python files in the following order: data\_storing.py, classifier.py (to produce the classifier.pkl file), then run app.py. If you already have access to classifier.pkl saved, then app.py can be run immediately.

Tkinter is a python library that is specifically used for creating graphical user-interfaces. It consists of several built-in GUI components such as buttons, labels and various menus that can be used to build desktop applications with a graphical interface. In the user-interface created below Tkinter was employed on multiple occasions. The main application window denoted as ‘Tk()’ was utilized to essentially create the main application window while ‘Button()’ allowed the creation of buttons with specified features like text and command functions. The creation of these widgets were then packed into the window from the ‘pack()’ feature to be arranged within the GUI. In the grand scheme of model deployment Tkinter played a crucial role in implementing user-interface and user-interaction within the usability aspect of the application. The model was exported to a binary file to be reusable without having to be re-trained each time the application was opened. All parts of the code was consolidated into reusable modules and functions for proper organization of the application. Individual testing was also performed which involved smaller functions and components to be tested to ensure the necessary output was attained. A performance test was also conducted to test the system in various conditions like window sizes and overall scalability of the system.

* 1. A screenshot of a computer

     Description automatically generated

Figure 23 Input csv file

* 1. A screenshot of a computer

     Description automatically generated

Figure 24 Raw Data Set csv file

* 1. A graph of a graph

     Description automatically generated

1. A screenshot of a computer

   Description automatically generated

Figure 25 Raw Data set

1. A screenshot of a computer

   Description automatically generated
2. A screenshot of a computer

   Description automatically generated

Figure 26 Cleaned up data set

# Participation Report

|  |  |
| --- | --- |
| Individuals Name | Respective Task |
| Anapayaan Pakerathan | Data Collection Written, Personal Data collection, Data visualization code, App GUI code, Visualization written, Model Deployment written, Demo Video |
| Hendrix Gryspeerdt | Data storing (code+written), Personal Data, Pre-processing(code+written), Feature extraction and normalization(code+written),Training the classifier(code+written), APP GUI code+plotting, Demo Video |
| James Dockrill | Proof-reading entire document |

# References

[1] “Your smartphone is a mobile lab.,” phyphox. Accessed: Apr. 07, 2024. [Online]. Available: https://phyphox.org

[2] “pandas.DataFrame — pandas 2.2.1 documentation.” Accessed: Apr. 07, 2024. [Online]. Available: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

[3] “sklearn.preprocessing.MinMaxScaler,” scikit-learn. Accessed: Apr. 07, 2024. [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[4] “scikit-learn: machine learning in Python — scikit-learn 1.4.1 documentation.” Accessed: Apr. 07, 2024. [Online]. Available: https://scikit-learn.org/stable/index.html

[5] “pickle — Python object serialization,” Python documentation. Accessed: Apr. 07, 2024. [Online]. Available: https://docs.python.org/3/library/pickle.html