

ELEC 292

Project Report

Group Number 30

Ryan Zietlow, 20347719, 21rjz3@queensu.ca

Alan Hu, 20352082, 21ah117@queensu.ca

Kai Young, 20370170, 22jky@queensu.ca

April 7th, 2024

Table of Contents

Introduction.....	3
Data Collection	3
Data Storing	3
Visualization	4
Pre-Processing.....	10
Feature Extraction and Normalization	10
Training the Classifier.....	11
Model Deployment	13
Participation Table	15
Bibliography	15

Table of Figures

Figure 1: X, Y and Z walking acceleration of each team member on a 3D scatter plot.	4
Figure 2: X, Y and Z jumping acceleration of each team member on a 3D scatter plot.	5
Figure 3: Absolute acceleration of as a function of time for Ryan's walking data.	5
Figure 4: Absolute acceleration of as a function of time for Alan's walking data.	6
Figure 5: Absolute acceleration of as a function of time for Kai's walking data.	6
Figure 6: Absolute acceleration of as a function of time for Ryan's jumping data.	7
Figure 7: Absolute acceleration of as a function of time for Alan's jumping data.	7
Figure 8: Absolute acceleration of as a function of time for Kai's jumping data.	8
Figure 9: Devices used to collect data for the experiment.	8
Figure 10: Positions the mobile device was in while the data was being collected.....	9
Figure 11: Confusion Matrix	12
Figure 12: ROC Curve	12
Figure 13: Accuracy of the classifier	13

Table of Tables

Table 1: Participation Report	15
-------------------------------------	----

Introduction

The goal of this project is to construct a desktop application that can distinguish between the action of walking and jumping. It should be taking accelerometer data along the X, Y and, Z axes from a CSV file and outputs a new CSV file that contains the new output data that labels it with walking or jumping.

The team followed a seven-step process to ensure the accuracy of the application. First, the team collected data for three minutes of walking and jumping with the mobile device in different locations to ensure all test conditions were considered. The team then organized and stored the data, plotted it to draw conclusions, pre-processed, normalized and extracted the features to ensure the model had a minimal level of noise so the model could be fitted to the best of its abilities, followed by training the classifier using StandardScalar and a maximum 10,000 iterations, and finally implemented the model into a GUI using the python library tkinter.

The team created a successful model, operating at an accuracy and recall of 100%.

Data Collection

The team used a mobile application called Phyphox to collect data. Phyphox is an app that collects data by using built-in sensors such as accelerometers, gyroscopes, and magnetometers and allows users to visualize and export data into various formats. The team collectively agreed to collect three minutes of both walking and jumping data, with the phone in three separate positions for each one-minute segment of the process. The phone started off in the collector's right hand, followed by the left pocket and then inside the back of their pants waste band. This occurred during both the walking and jumping data collections. Once the data was collected each member exported it into a CSV file and emailed it to Alan to put into PyCharm. To differentiate between the walking data and jumping data two different labels were used, 0 for walking and 1 for jumping.

The team encountered a couple minor setbacks throughout the collection process. The first being fatigue when attempting to jump for three consecutive minutes. To solve this issue the team paused the collection after each one-minute segment to regain stamina to continue with the jumping. The second issue was members used different versions of the collection software. Ryan and Kai collected their acceleration “with g” and Alan first completed it “without g”. After realizing the issue, it was swiftly fixed, and all members collected data “with g”.

Data Storing

The team took multiple steps to correctly store the raw data. First, a function was made to label the raw data. It takes a CSV file with unlabeled data as the input, reads the CSV into a data frame and adds an extra column assigning a value of 0 if it's walking or 1 if it's jumping. It then saves the labelled data frame back into a new CSV file. Next the data must be shuffled. A shuffle data function that takes in the labelled CSV as input and will output a shuffled CSV file. The function reads the labeled data and stores it into a data frame. Then, a new column called “window” is created by dividing the “Time (s)” column by 5 and applying floor division. This creates groups

of data in 5 second windows, for example data points collected between 0 and 4.99s are put into the same window. Next, the unique windows are shuffled to remove any time-based patterns that could cause bias and a mapping between the original windows and the shuffled windows is created and is applied to the window column of the data frame. The data frame is then sorted to ensure the data within the window is in the correct chronological order, but the windows are shuffled and randomized. This data frame is then saved again into a new CSV file. The next step is to create a combine function that merges the data from all three participants into single files for walking and jumping. After this, it is time to split the data set into training and testing by defining a split dataset function. It loads a CSV into a data frame and then splits the data into 90% training and 10% testing. It is then converted back into a CSV. After the data is correctly labeled, shuffled, combined, and split, they are loaded into HDF5 files.

Visualization

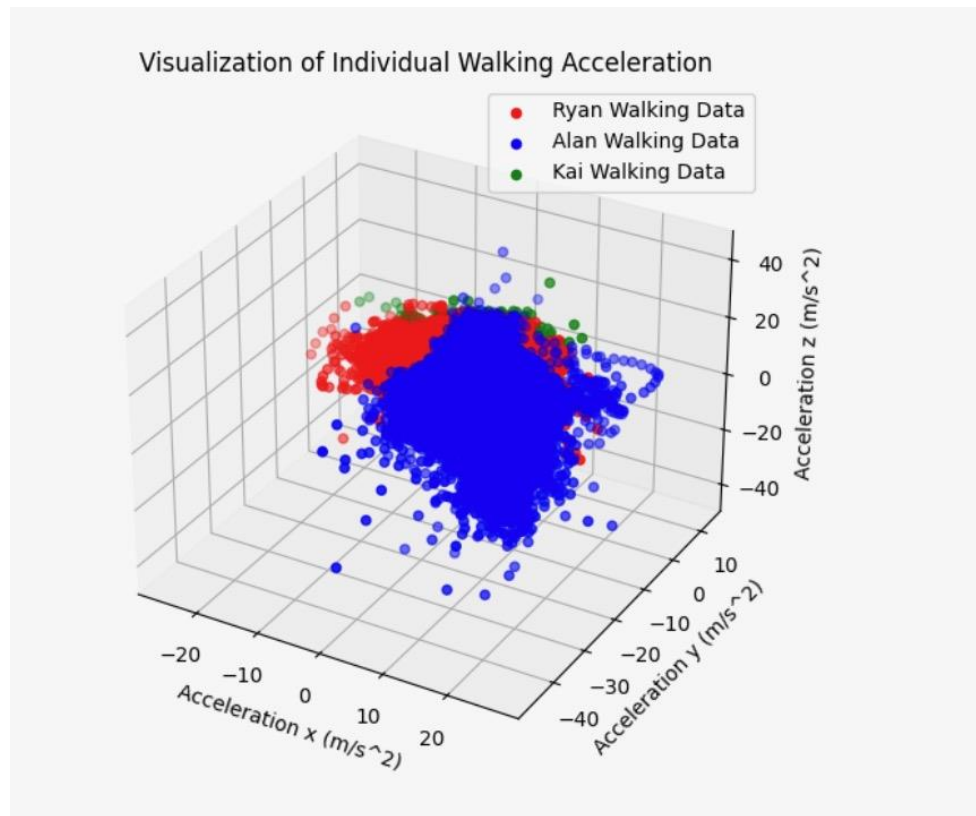


Figure 1: X, Y and Z walking acceleration of each team member on a 3D scatter plot.

Visualization of Individual Jumping Acceleration

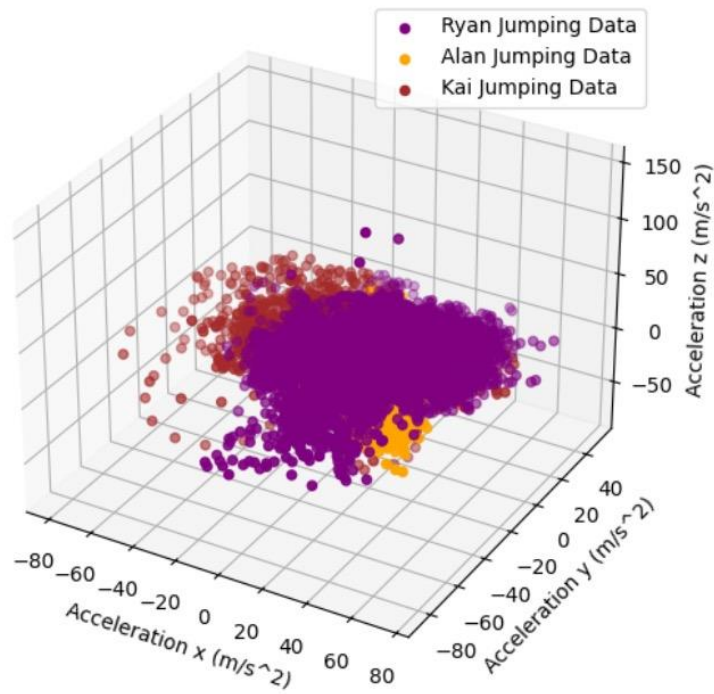


Figure 2: X, Y and Z jumping acceleration of each team member on a 3D scatter plot.

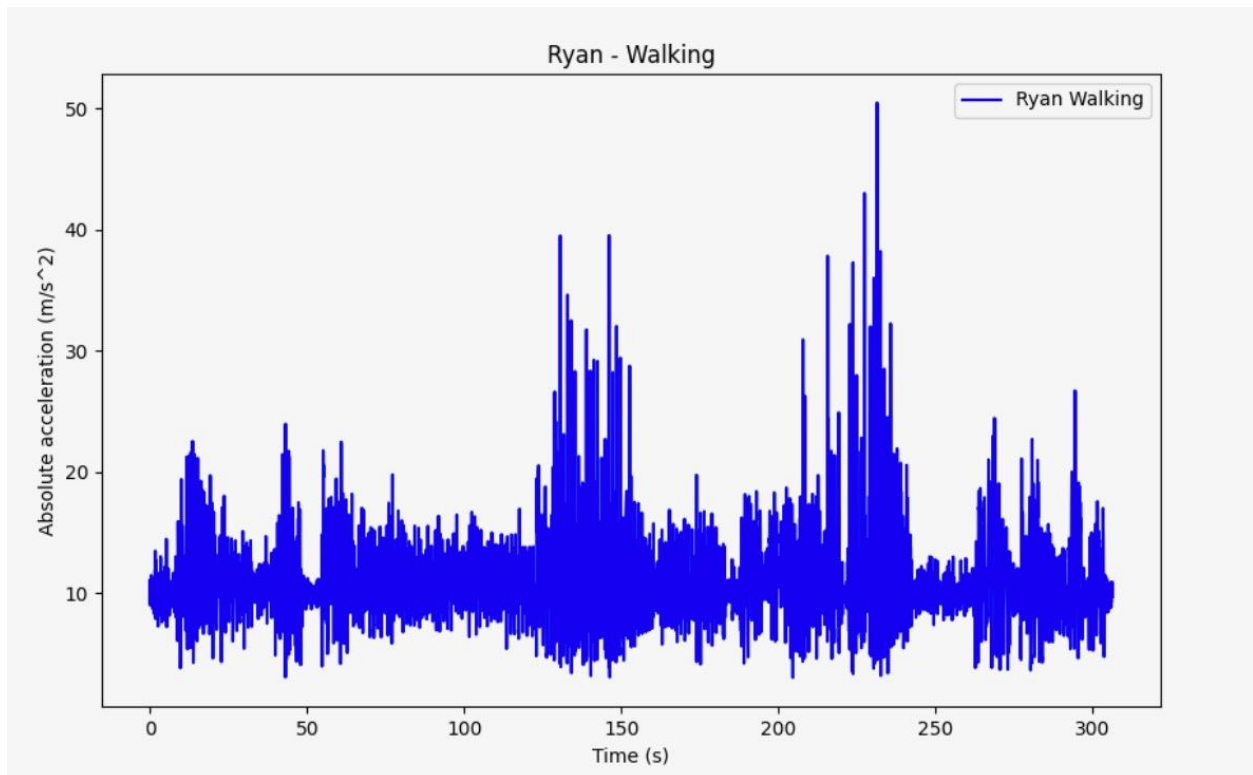


Figure 3: Absolute acceleration of as a function of time for Ryan's walking data.

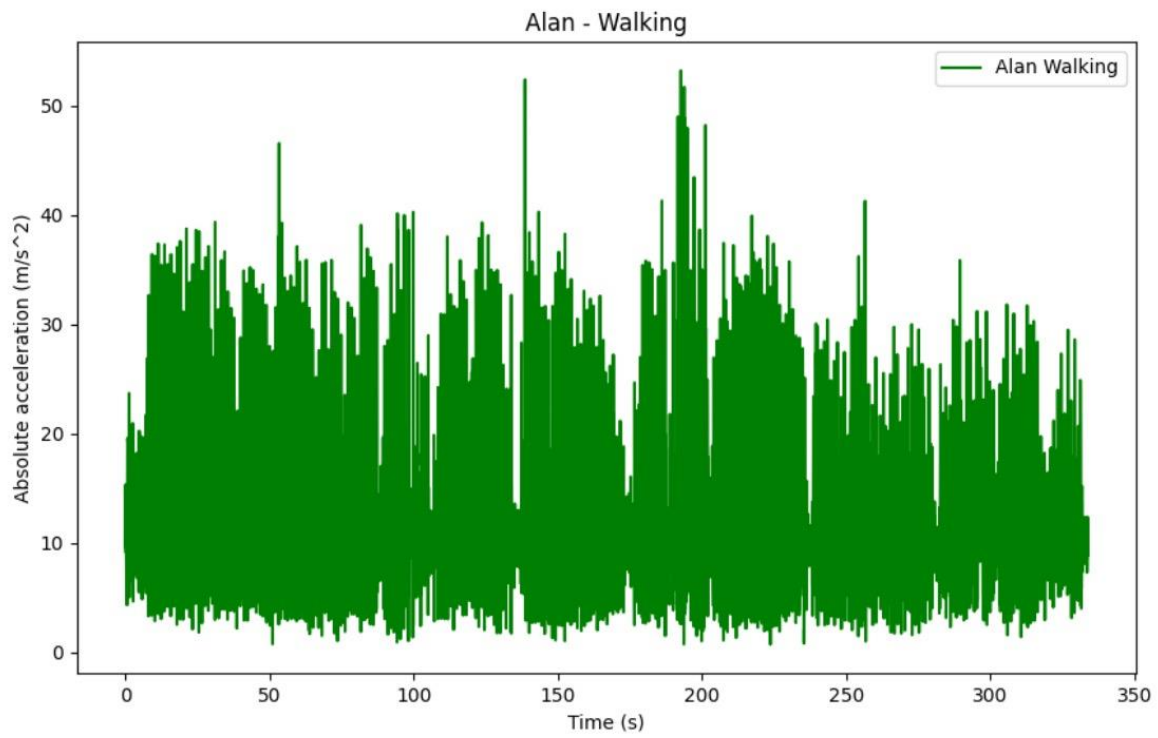


Figure 4: Absolute acceleration of as a function of time for Alan's walking data.



Figure 5: Absolute acceleration of as a function of time for Kai's walking data.

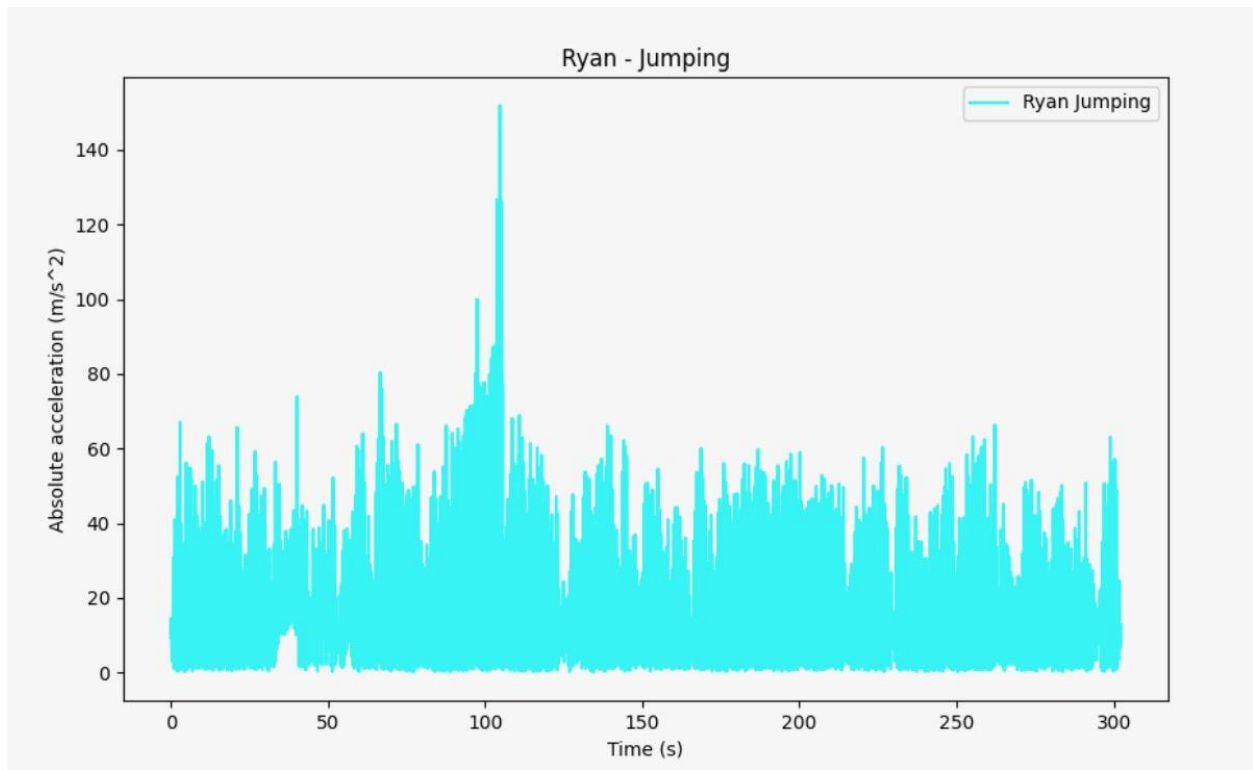


Figure 6: Absolute acceleration of as a function of time for Ryan's jumping data.

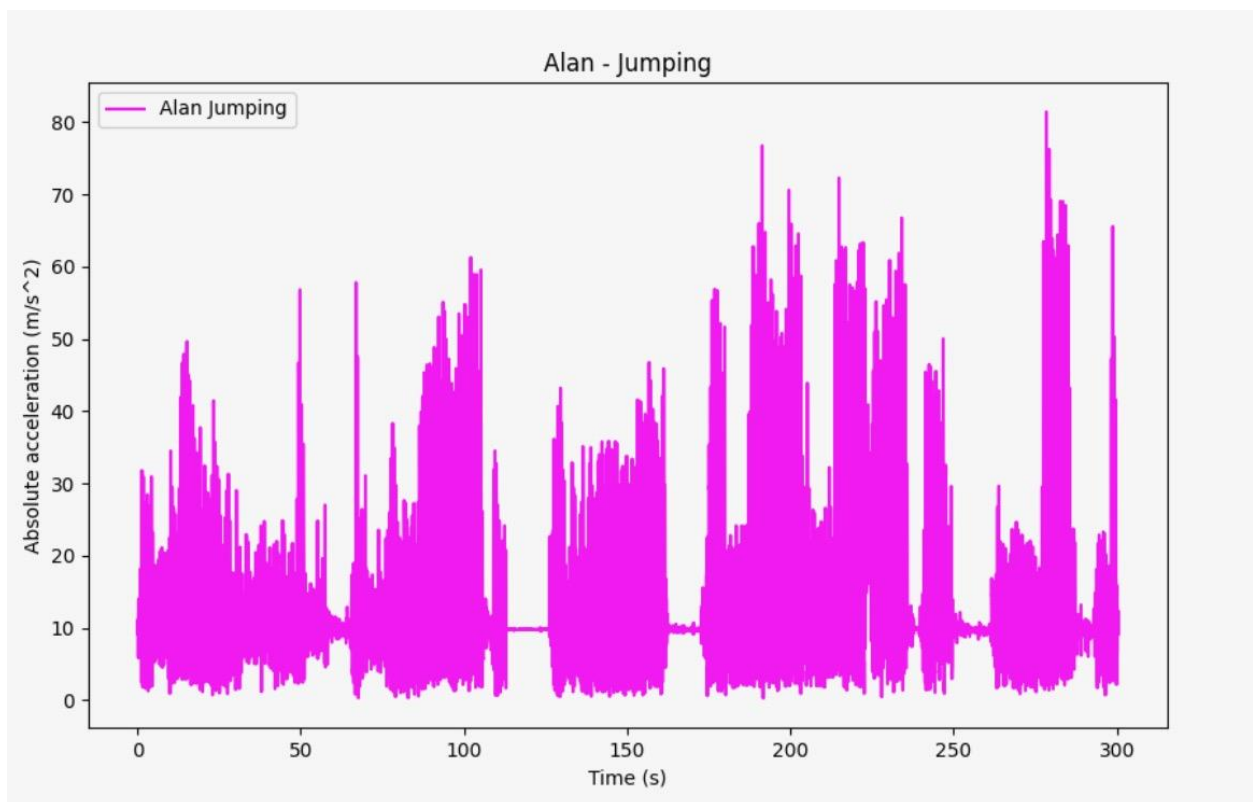


Figure 7: Absolute acceleration of as a function of time for Alan's jumping data.

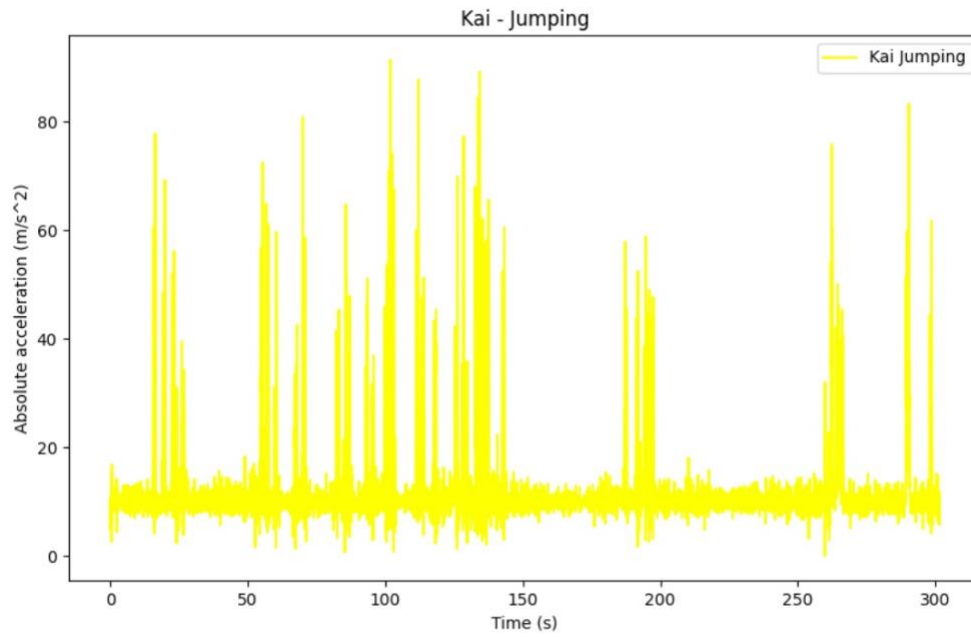


Figure 8: Absolute acceleration of as a function of time for Kai's jumping data.

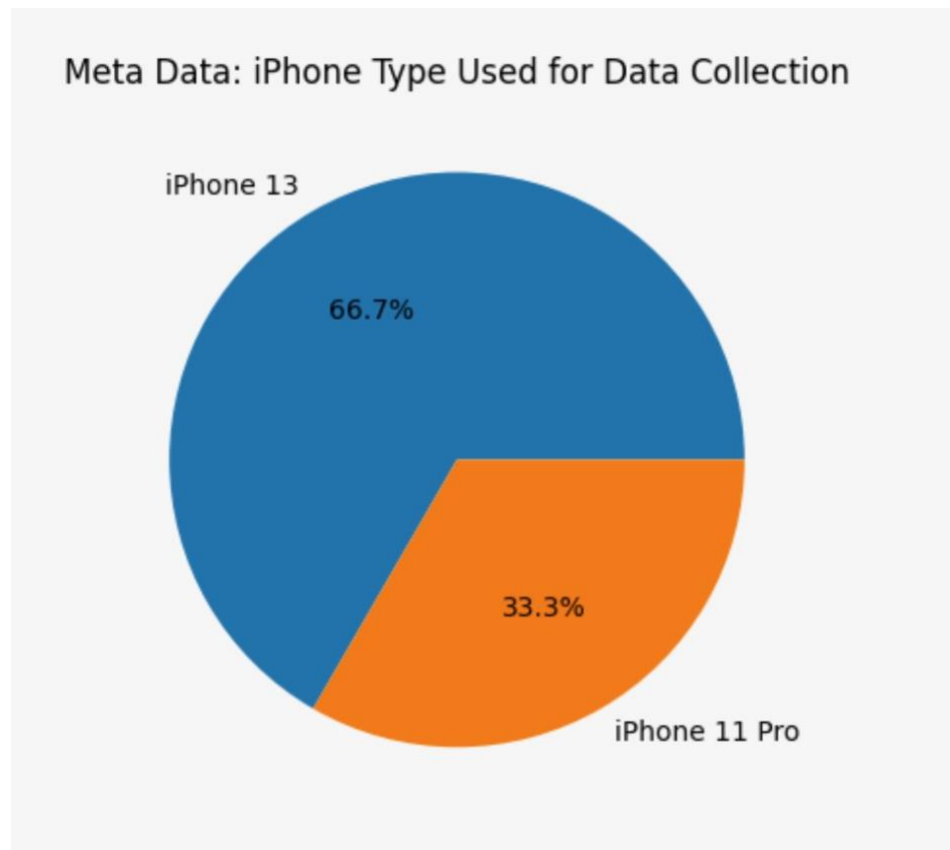


Figure 9: Devices used to collect data for the experiment.

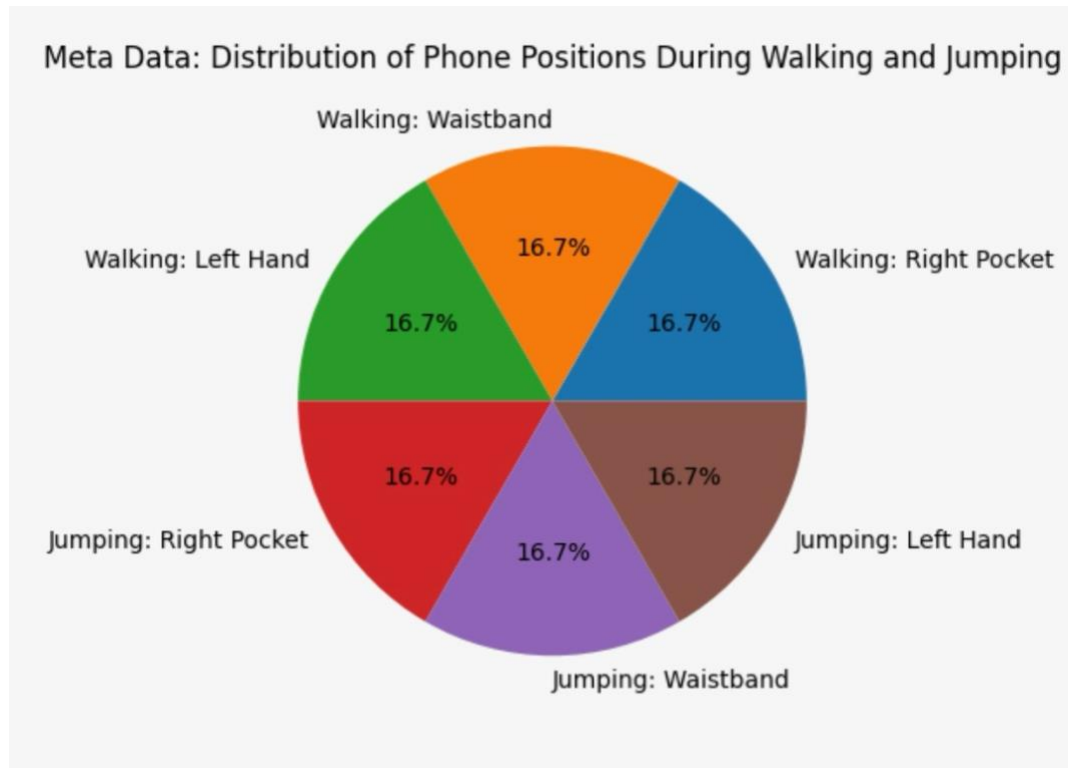


Figure 10: Positions the mobile device was in while the data was being collected.

Figure 1 and Figure 2 display the acceleration data in the X, Y and Z direction for each team member while walking and jumping. The team learned a valuable lesson from the plot, that is, each user has their own pattern in the data clusters, most likely relating to different unique cadences and movement styles. It also is very hard to interpret as the data is very clustered together, although outlier stand out in each direction. If the data were to be recollected again, the team would ensure that the data would be collected in the same environmental conditions to avoid any unnecessary outlier data points.

Figure 3, Figure 4, Figure 5, Figure 6, Figure 7, and Figure 8 display the absolute acceleration as a function of time. Each participant has their own graph for walking and jumping respectively. The lessons learned from this graph is very evident as it did a good job at visualizing the high peaks of intensity between walking and jumping. Furthermore, the data plotted along the graphs all look similar which gives the team confidence that the data was correctly collected. To further improve if we were to collect data again, the team would be to collect the data in a controlled environment as walking downstairs has created visible outliers in the data.

Figure 9 and Figure 10 display meta data for the project. Figure 9 displays what type of mobile device the data was collected on, and Figure 10 displays the positions that were used to collect data for each activity.

Pre-Processing

For the pre-processing step, the team had to reduce the noise to ensure that when the classifier was going to be trained on the data, it was as accurate possible. To filter the noise, a rolling mean (simple moving average) with a window size of 5 was applied to each column of the dataset, excluding the first and last columns which got removed. The rolling mean helps smooth the data and reduces rapid fluctuations to make trends more evident for the model to predict. A window size of 5 was chosen to keep important features where the data hit high and low values but reduced any major unnecessary noises. If it was too large, the data would be oversmoothed and you would lose important data points but if it was too small the noise will not be filtered out and the model will have a hard time finding trends.

The first and last columns from the dataset are removed because they were the labels and the number of the data points which were both unnecessary to include. The method `sma5.drop()` method drops any rows that contain 'NaN' values which would make it impossible to calculate the rolling mean if they were in the dataset.

Feature Extraction and Normalization

The feature extraction process is crucial in ensuring the data is as useful as possible to help the model differentiate between walking and jumping. The features that were extracted include: mean, max, min, median, standard deviation, skew, kurtosis, variance, and the sum. They were each extracted for the X, Y and Z accelerations.

The team created a features function that first divided the dataset into segments of 500 data points each, allowing us to analyze the data in big groups to form meaningful conclusions. Then for each segment of data the nine features were extracted for each direction of acceleration.

These specific features were extracted for a multitude of reasons. Firstly, the mean is the average value and gives a general idea of the magnitude. The mean captures the intensity of the exercise, so if the mean has a high value, it could conclude that work is being done intensely meaning the activity could be jumping. The maximum value is the highest value in the dataset, and it allows the model to learn where the peak of the data is which is again useful to see how intense the exercise is. The min is the lowest value in the segment, and it can help identify when intensity is at its lowest meaning it the exercise could be walking. The median is the middle value of the data in the segment, and it helps smooth any outliers if there is a very high or low data point in the segment. The standard deviation displays how far away the value is from the average. It could help identify a sudden high or low intensity movement if it is much higher or lower than the mean. The skewness measures the asymmetry of the data distribution around the mean. If the skew is positive or negative it could indicate any biases in the movement such as leaning in one direction. The kurtosis measures the tailedness of the distribution of the data points. If it is high, it could display that the data deviates from the mean very often, meaning a movement could be giving a sudden start or stop. The variance is the deviation squared. It helps analyze how far the data deviates from the average. The sum is the total of the data points in a segment. It can be useful if there is a lot of total motion over the segment. Overall, these features can display distinct patterns in the data that can assist in predicting the activity that is occurring.

There have been previous studies regarding the motion detection of activities using sensors such as “Smartphone sensors-based human activity recognition using feature selection and deep decision fusion” by Yijia Zhang and Xiaolan Yao. These studies also use the approach of segmenting the data and extracting very similar features across each axis of acceleration to help capture information to train the model more accurately. [1]

For the normalization process, the team used the standard scaler from the sklearn library to scale the extracted features and ensure that each feature contributes equally to the model’s performance. First an instance object of the StandardScaler was created to standardize the features. Next the scaler has to be fitted, so the `fit_transform()` method is called on the dataset to compute the mean and standard deviation of each feature in the dataset. The model will use the mean to subtract it from each data point to find the center it, and the model will use the standard deviation to scale the data. After the scaler is fitted, the `fit_transform()` method will transform the dataset, essentially standardizing the distribution to have a mean of zero and a standard deviation of one to ensure that the features are each weighed equally in the model. After this is completed, the data is in the form of an array, and it is converted back into a data frame and the labels are added to classify what task is occurring. The data frame is then written to a new CSV file so it can be used in the classifier in task 6.

StandardScaler was chosen because a handful of machine learning models work better when features are weighed equally, and it reduces the impact of outliers on the dataset.

Training the Classifier

The first step of training the classifier was loading the data into separate training and testing datasets. It is followed by data concatenation where both training and testing datasets were concatenated into singular data frames for both walking and jumping respectively. The next step is to separate the features from the labels using `iloc[:, -1]` to select all columns except the last to ensure that the model is not training with the labels on the data. Next, a machine learning pipeline was created using `make_pipeline` which uses the StandardScale to ensure all features are contributing equally and LogisticRegression to differentiate between walking and jumping. Following this, the model is then fit on the training data using `clf.fit(combinedTrainingDF, trainLabel)` which trains the classifier on the features. 10000 was the maximum number of iterations that was used to ensure the model converged on an accurate outcome. This was used to give the model ample time to find the solution.

The accuracy, recall, confusion matrix, ROC curve and AUC were used to evaluate the classifiers performance. The accuracy is a good tool as it measures the overall accuracy of the model. The recall is sufficient as it measures the proportion of how many of the positive cases that were predicted by the model are positive. The confusion matrix helps visualize the true positives, true negatives, false positives, and false negatives. The ROC curve does a good job at visualizing how well the model distinguishes between positive and negative. The AUC measures the area under the ROC curve and provides a value that tells you about the accuracy of the model.

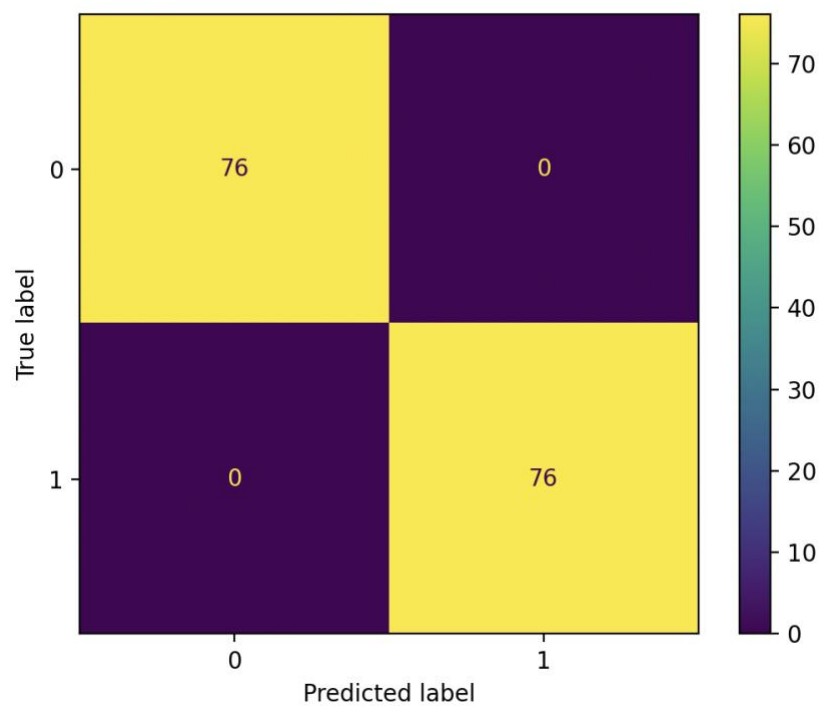


Figure 11: Confusion Matrix

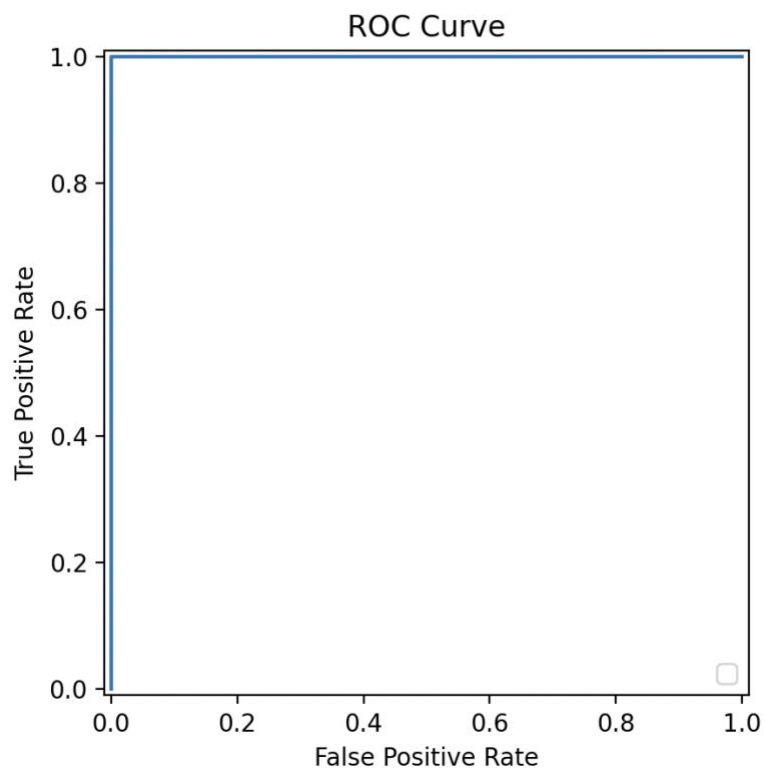
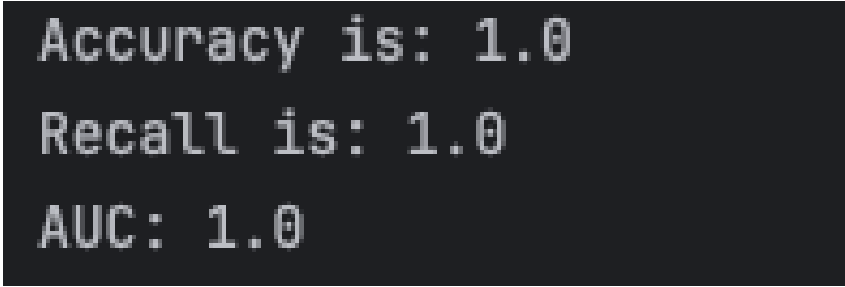


Figure 12: ROC Curve



```
Accuracy is: 1.0
Recall is: 1.0
AUC: 1.0
```

Figure 13: Accuracy of the classifier

Model Deployment

To deploy the model into an application, the team used the library Tkinter to construct a graphical user interface. It takes in an input file and outputs a labelled CSV for each window in the file and generates a plot to represent the output. The first step was to call a `noiseFilter()` function to apply a rolling mean and drop 'NaN' values. Then the data frame was segmented groups of every 500 data points, and each group had the features calculated on them. Next, `RobustScaler` was called to scale the features and returned a new data frame of the scaled features. The next step was to call the noise filtering and feature normalization functions on the data frame, uploads it into the model to predicts the labels and saves it to a CSV file. Then an import function is created to allow the user to input a CSV file to get it read to a data frame for it to be processed.

Unfortunately, the model was not able to be properly deployed onto the GUI due to feature names not being matched between the training model and prediction phase. As new data is inserted for an attempt to make a prediction, it looking at feature names the model hasn't seen before. The team spent dozens of hours attempting to fix this error, yet they had no luck.

As you can see in Figure 14, the GUI has a visual on the front that was generated by an AI Image generator. Then there is a button down below that allows the user to insert a file to get predicted on, and an output file and graph would be sent back in return.

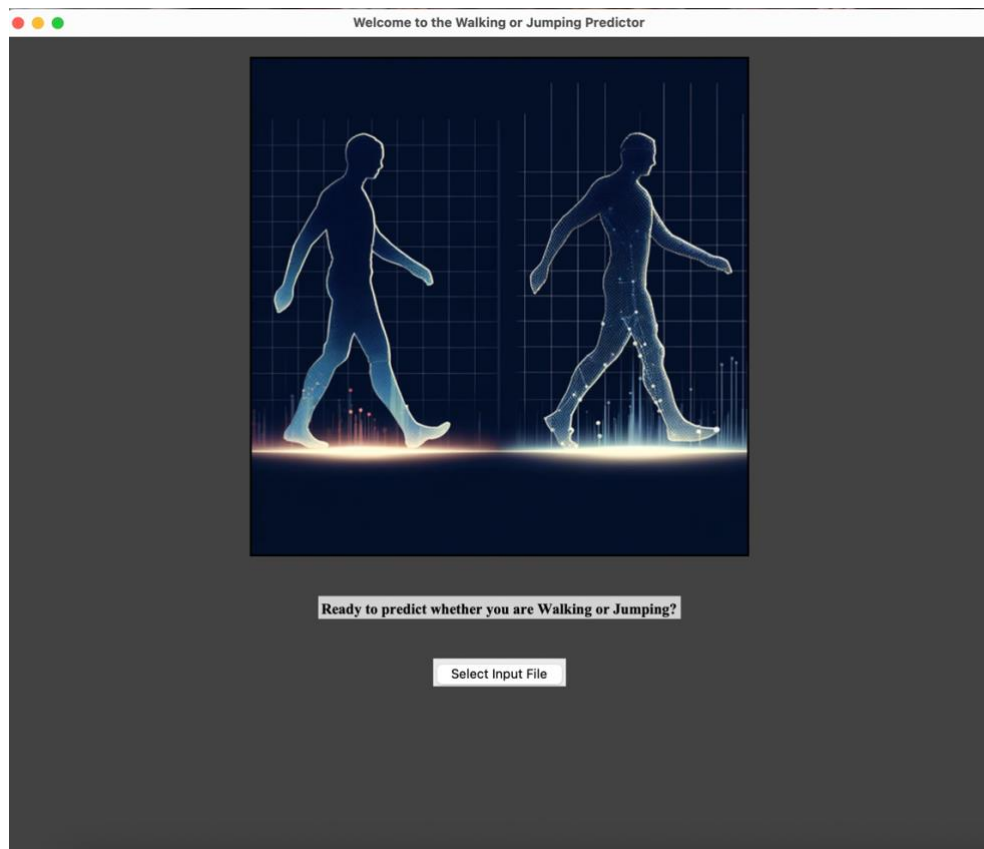


Figure 14: The user interface of the model

```

selected file: /Users/ryan/Desktop/Final Project 202/stupid/stupid_jumping.csv
Exception in Tkinter callback
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.12/Lib/python3.12/tkinter/_tkinter.py", line 1962, in __call__
    return self.func(*args)
  File "/Users/ryan/Desktop/Final Project 202/stupid.py", line 89, in importFile
    df = process_data(df)
  File "/Users/ryan/Desktop/Final Project 202/stupid.py", line 75, in process_data
    df = model.predict(df)
  File "/Library/Frameworks/Python.framework/Versions/3.12/Lib/python3.12/site-packages/sklearn/pipeline.py", line 602, in predict
    Xt = transform.transform(Xt)
  File "/Library/Frameworks/Python.framework/Versions/3.12/Lib/python3.12/site-packages/sklearn/utils/_set_output.py", line 295, in wrapped
    data_to_wrap = f(self, X, *args, **kwargs)
  File "/Library/Frameworks/Python.framework/Versions/3.12/Lib/python3.12/site-packages/sklearn/preprocessing/_data.py", line 1043, in transform
    X = self._validate_data(
  File "/Library/Frameworks/Python.framework/Versions/3.12/Lib/python3.12/site-packages/sklearn/base.py", line 468, in _validate_data
    self._check_feature_names(X, reset=reset)
  File "/Library/Frameworks/Python.framework/Versions/3.12/Lib/python3.12/site-packages/sklearn/base.py", line 535, in _check_feature_names
    raise ValueError(message)
ValueError: The feature names should match those that were passed during fit.
Feature names unseen at fit time:
- kurtosis absolute
- kurtosis x
- kurtosis y
- kurtosis z
- max absolute
- ...
Feature names seen at fit time, yet now missing:
- 0
- 1
- 10
- 11
- 12
- ...

```

Figure 15: The error the team could not figure out

Participation Table

Table 1: Participation Report

Project Task	Ryan	Alan	Kai
Data Collection	X	X	X
Data Storing	X		
Data Visualization		X	
Pre - Processing	X	X	X
Feature Extraction	X		X
Training the Classifier		X	X
Model Deployment	X	X	X
Demo Video	X	X	X
Report	X	X	X

Bibliography

- [1] Y. Zhang, 15 December 2022. [Online]. Available:
<https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/cps2.12045>.