Harry Redman

Registration number 100340391

2023

Using Time Series Classification in the Gym with smartphones to gamify and encourage exercise

Supervised by Jason Lines

University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

This report evaluates the performance of various classification algorithms on a dataset of gym exercises recorded using time series data from the accelerometer and gyroscope sensors of a smartphone device. The primary objective is to determine if you can accurately classify gym exercises based on the sensor data collected.
The project compares a number of common machine learning algorithms such as Random Forest, k-Nearest Neighbours with Euclidean Distance and Naïve Bayes, as well as some time series specific algorithms, including Rocket and Time Series Forest, on a range of different variations of the dataset. These variations include different attributes such as using univariate data from a single axis or using multivariate data by combining data from multiple axes. Prior to analysis, the time series data is pre-processed to use same-length normalised time series data of 10 seconds.
The results of this project show that when using multivariate time series data that uses both the accelerometer and gyroscope data, an accuracy of 99.2% can be achieved using the Rocket classifier.

Acknowledgements

I would like to thank my supervisor Dr Jason Lines for his continuous support and advice throughout this project as it would not have been possible without him. I would also like to thank Barnaby Johnson for his contribution to the data collection of this project.

Contents

1. Introduction **5**

2. Background **6**

2.1. Time Series Classification 6

2.2. Comparing classifiers in Time series 6

2.3. Related Work 7

3. Preparation **9**

3.1. Data Collection 9

3.1.1. Chosen Gym Activities 9

3.1.2. Sensors used 10

3.1.3. Data Collection Methodology 12

3.2. Machine Learning Classifiers 14

3.2.1. K Nearest Neighbours 15

3.2.2. Decision Tree 16

3.2.3. AdaBoost 16

3.2.4. Random Forest 17

3.2.5. Naïve Bayes 17

3.2.6. BOSS 17

3.2.7. Multi-Layer Perceptron 17

3.2.8. Time Series Forest 18

3.2.9. Rocket 18

3.3. Evaluation Metrics 19

3.4. Data Preprocessing 20

3.5. Dataset Variations 22

3.6. Prototype Experiment 23

4. Implementation and Evaluation **25**

4.1. Experimental Plan 25

4.1.1. Experiments Performed 25

4.1.2. Generating Data Splits 27

4.1.3. Experiment Implementation 28

4.2. Analysis of Results 29
4.2.1. 8 Class gym exercise problem 29
4.2.2. Bodyweight or Weighted Exercises 33
4.2.3. Multiple Sensors 35
4.2.4. Univariate or Multivariate 36
4.2.5. Person Independent 38
4.3. Real-time exercise classification 40

5. Conclusion and Future Work 41
5.1. Conclusion 41
5.2. Future Work 42

References 43

A. Appendix 45

1. Introduction

This project will focus on classifying a range of common gym exercises using Time Series Classification (TSC), a subfield of machine learning. The main goal of this project is to determine if Time Series Classification can be used to accurately predict the following gym exercises: barbell squats, barbell bench press, barbell deadlifts, barbell military press, press ups, pull-ups and sit-ups using accelerometer and gyroscope sensor data recorded from an Android smartphone. The sensor data recorded will then be processed and formatted to generate multiple dataset variations of the gym exercises recorded. Which will be used to train and test a range of machine learning models, including some generic algorithms as well as time series specific classifiers. This will be done to answer the five main research questions that are going to be investigated throughout this report, which are:

1. Can Time Series Classification algorithms be used to accurately classify between 8 different gym exercises?
2. Does using both the accelerometer and gyroscope sensors together increase model performance, over using just one of the sensors?
3. Is multivariate data better for classifying gym exercises than univariate data?
4. Are weighted gym exercises or bodyweight exercises easier to classify using Time Series Classification algorithms?
5. Are the classifiers person dependent or will a model built on one person's data be able to accurately classify the gym exercises of another person?

To answer these research questions and identify which algorithm performs best for this TSC problem, each of the models developed by this project will be evaluated and compared using various evaluation metrics. The results from the Time Series Classification aspect of this project will then be used to develop a prototype app that predicts gym exercises in real-time that aims to encourage more people to participate in physical activity by tracking their exercise.

The motivation for this project is due to rising obesity rates in the world population, as well as growing concerns about health risks caused by living a sedentary lifestyle such as Type 2 diabetes or stroke. According to Wang et al. (2011), an additional 11 million people in the UK are projected to be obese by 2030 compared to 2010. Causing an annual increase in healthcare costs of £1.9-2 billion for obesity-related health issues. This is a significant issue that must be addressed, and one of the most effective ways to reduce obesity is through increased physical activity. Therefore, through my project, I hope to be able to create a classifier model which can accurately classify gym exercises, and then be used to track exercise and potentially gamify exercises to encourage more people to increase their physical activity.

2. Background

2.1. Time Series Classification

A time series is a collection of ordered data points, that is typically collected over a period of time and has evenly spaced intervals between the data points, as defined by Lines et al. (2012). Time Series Classification is a form of machine learning problem, that looks at predicting a class for a given time series based on a labelled training set of other time series. Ruiz et al. (2021) describes that for a Time Series Classification problem a $\boldsymbol{n}$ set of time series is required: $T = \{t_1, t_2, \ldots, t_n\}$. There are two types of TSC problems: univariate and multivariate. In a univariate problem, each instance of time series t_i only has a single series of $\boldsymbol{m}$ ordered and real-valued data points: $t_i = \langle t_{i,1}, t_{i,2}, \ldots, t_{i,m} \rangle$ and a given class label c_i. Alternatively, there is a multivariate TSC problem, where each instance of time series has multiple dimensions: the time series is a list of arrays over $\boldsymbol{d}$ dimensions and $\boldsymbol{m}$ observations, $X = \langle x_1, x_2, \ldots, x_d \rangle$, where $x_k = \{x_{1,k}, x_{2,k}, \ldots, x_{m,k}\}$. Human Activity Recognition (HAR) is an example of a multivariate Time Series Classification problem and is the category the problem of this project falls under. Bagnall et al. (2018) defines HAR as the problem of predicting an activity based on accelerometer and/or gyroscope data.

2.2. Comparing classifiers in Time series

Dau et al. (2019) presents a standard method for evaluating and comparing the performance of different classifiers on a time series problem. Classifier performance can be compared using one of two general methods: either the classifier's prediction ability or the probability estimates. When measuring a classifier through its ability to predict, the most common evaluation metric used is accuracy. However, this metric does not take into account class imbalances and therefore other metrics that can measure these imbalances would be more informative for imbalanced class datasets. The most common evaluation metrics used for a binary classification problem are accuracy, balanced accuracy, sensitivity, specificity, recall and the f statistic. Dau et al. (2019) mentions that however, for multi-class problems accuracy and balanced accuracy are considered enough to evaluate the predictive ability of a classifier. Some classifiers produce probability estimates and for this, metrics such as negative log-likelihood or area under the receiver operating characteristic curve (AUROC) can be used.

When comparing classifiers on a small and single dataset, two main problems can occur: the temptation to cheat results by setting the classifier parameters to optimise the test data and that small differences in performances can seem magnified. Dau et al. (2019) explains these problems can mostly be overcome by combining the train and test data, re-sampling the data set multiple times and then averaging the test accuracy. Several conditions must be met if this method is used:

- The default train test splits should always be included as the first re-sample
- The re-samples for each classifier must be the same
- The re-samples should keep the same train and test sizes
- The re-samples should be stratified to maintain the original class distribution

Once you have an evaluation metric to measure the performances of different algorithms a significance test should be done to test if there is a significant difference in the performances of the classifiers. Dau et al. (2019) outlines two hypothesis test's that can be used in a time series problem experiment: a paired two-sample t-test to determine if there is a significant difference in mean accuracies or a Wilcoxon signed-rank test to see if there is a difference in median accuracies.

2.3. Related Work

In order for me to select the best methods to use for my own work, it is important to analyse other similar work to identify the strengths and limitations of the methods they used. Time Series Classification for Human Activity Recognition problems has already produced promising results, highlighting its potential for real-world use and giving me an insight into the sort of results I could expect from my own time series problem.

A paper by Nurwanto et al. (2016) looked at detecting light sport exercises using the accelerometer data collected from a smartwatch and smartphone. The exercises they recorded data on for this project were push-ups, sit-ups, squat jumps and walking. In total, the number of data instances they used was 25, with 15 of these being used for a training set and the other 10 being used as the test set. For this project, they used a k-Nearest Neighbours (kNN) classifier with a time series specific distance algorithm called Dynamic Time Warping, which was tested on the following k values: 1, 3, 5 and 7. To determine the best k value to use for the classifier a 5-fold cross-validation was performed on the training dataset to find the value that produced the best performance. To evaluate the performance of the different models, the following evaluation metrics were calculated; accuracy, sensitivity and specificity. The results of the cross-validation on the training set found that the kNN classifier with a k value of 3 performed best and when evaluated on the test data achieved an accuracy of 76.67% for push-ups, 80% for sit-ups and 96.67% for squat jumps. Their project concluded that kNN with Dynamic Time Warping can be used to classify light sport exercises, which suggests that it would be a good algorithm to use for my Time Series Classification problem.

Another paper that focused on Human Activity Recognition was conducted by Khan et al. (2020), in this paper they focused on predicting gym exercises with smartphone sensors in a real-world setting. Three separate smartphones were used in this experiment, that were positioned on the arm, the belly and the leg. They then recorded the accelerometer and gyroscope readings from each smartphone to capture the data for each exercise to create their dataset. In this experiment they used 14 gym exercises, some of the exercises included were; barbell squat, decline close grip bench press and seated barbell shoulder press. For each exercise, two sets of ten repetitions were recorded and in order to reduce the noise in their data, they removed the first and last three seconds of each exercise's data collection. As a result, each set of an exercise produced 32 seconds of data. Using a 4-second sliding window technique, they then extracted the mean, standard deviation, maximum, and minimum features from this data.

The classifiers used in this study were all common machine learning algorithms that were implemented using WEKA. The three classifiers used were kNN with Euclidean Distance, Naïve Bayes and a j-48 Decision Tree. To train and test their models an 80/20 train-test split was used.

Comparing the results of all the algorithms when the smartphone was located on the arm and trained using the data from both the accelerometer and the gyroscope sensors individually, as well as both sensors combined. Showed that the kNN model had an accuracy of 95.87% for accelerometer data, 91.83% for gyroscope data, and 98.04% for both sensors together. While the Decision Tree classifier had an accuracy of 87.73% for the accelerometer, 80.23% for the gyroscope, and 90.63% for both sensors together. Finally, the accuracy of the Naïve Bayes classifier was only 62.49% for the combined sensors, while the accelerometer was just 52.61% and the gyroscope 31.39%.

The results of their research showed the best classifier to use for predicting exercises in the gym when the smartphone was placed on the arm was kNN, followed by a Decision Tree and then Naïve Bayes. One notable thing was that using both sensors resulted in better accuracy for all algorithms compared to when just using a single sensor. This suggests that using multiple sensors could improve the outcomes of my time series problem by capturing more data about an exercise. The location of the phone used to gather the data is another factor that is highlighted, showing that the location can have an impact on the outcomes and the data that is recorded. Therefore, for this problem, I will choose one location to record all my data.

3. Preparation

3.1. Data Collection

3.1.1. Chosen Gym Activities

For this project, I needed a time series dataset of gym exercises that I could run experiments on. Before I could conduct any experiments, I needed to record my own dataset of different gym exercises that I would then be able to use. The gym activities that I chose to collect data on can be divided into two main categories of exercise:

- Bodyweight exercises - which can be performed without any other equipment and use the individual's own bodyweight as a way to increase their strength or endurance.
- Weighted exercises - exercises that require weighted equipment such as barbells or dumbbells, which increases the difficulty of performing an exercise as a way to increase the individual's strength or endurance.

In total for this project, 8 different gym exercises were chosen for my data collection, these were: barbell squat, barbell bench press, barbell deadlift, barbell military press, sit-ups, press-ups, pull-ups and finally a stationary/null class that was selected for the real-time classification aspect of this project, which will be used to detect if an exercise was not being performed. These activities were chosen because all of them are core, staple exercises that work out all the major muscle groups in the body and would result in doing a full-body workout. Furthermore, the majority of gym-goers are aware of these exercises and how to perform them.

All the activities selected have enough range of movement so that when recorded the exercises should be able to be detected by the sensors and capture any characteristics of the movement. However, some of the gym exercises selected have similar movement paths such as military press and bench press. With the former requiring you to push a weight vertically above your head while standing and the latter pushing a weight away from your chest vertically while lying flat. The choice of selected exercises should allow for an interesting comparison into if you can distinguish between the different movements using Time Series Classification.

3.1.2. Sensors used

Trying to classify the gym exercises an individual is performing would be a HAR time series problem as mentioned in Section 2.1. Therefore, to collect the data for this project I will use both accelerometer and gyroscope sensors to record the exercises for the dataset. Both of these sensors were chosen because the information that each sensor captures is different and should give us more characteristics about the activity being recorded. Helping the algorithms distinguish between the activities being performed.

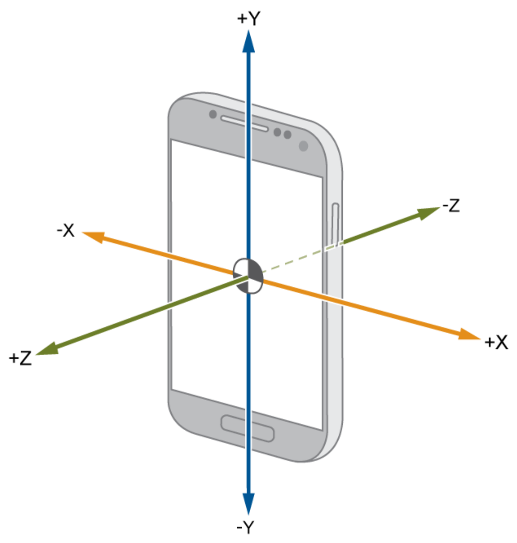


Figure 1: A diagram showing the accelerometer axes of a smartphone, illustrated by MathWorks (2023a)

An accelerometer sensor measures the acceleration of the device on three given axes. These three axes are: the *x-axis* (which measures lateral acceleration), the *y-axis* (that measures vertical acceleration) and then the *z-axis* (which measures the forward and backward acceleration of a smartphone). A demonstration of the 3 axes' movements for the accelerometer sensor in a smartphone can be seen in Figure 1.

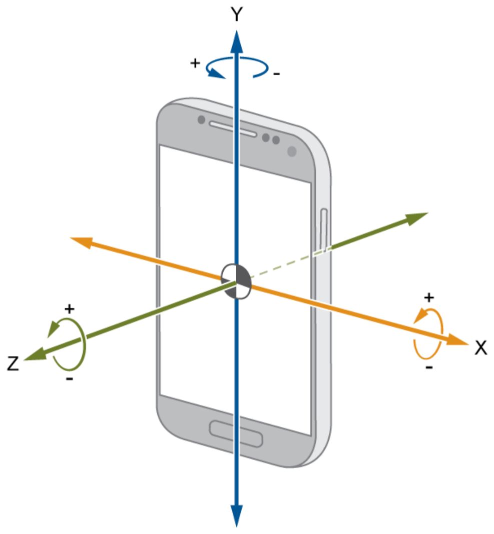


Figure 2: A diagram showing the gyroscope axes of a smartphone, illustrated by MathWorks (2023b)

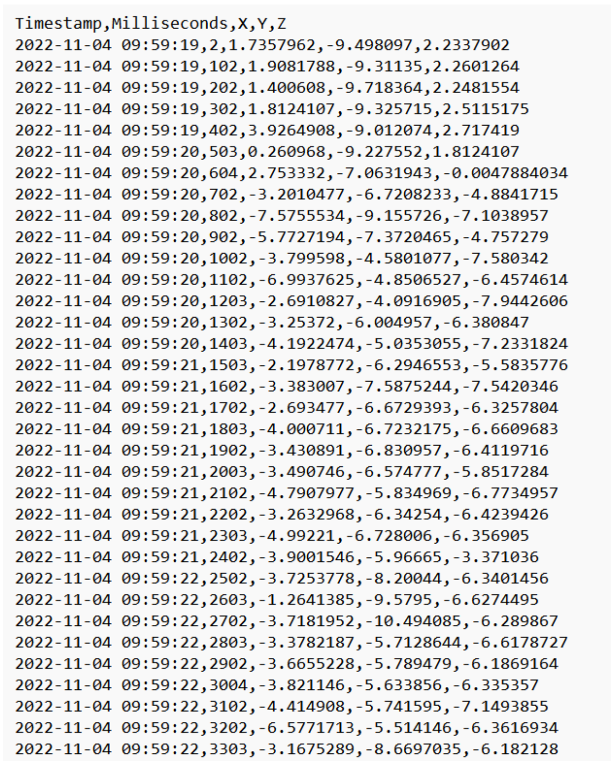
The gyroscope sensor like the accelerometer has three axes. Instead of measuring acceleration along the three axes, a gyroscope measures the angular velocity, which is the rate at which a device rotates around the three axes. The *x-axis* of a gyroscope measures the rate of rotation around the horizontal axis, the *y-axis* measures the rate of rotation around the vertical axis, and the *z-axis* measures the rotation of the forward and back axis. A diagram of how these gyroscope movements are captured is shown in Figure 2.

3.1.3. Data Collection Methodology

To collect my data, I decided on using a smartphone since almost all smartphones have built-in accelerometer and gyroscope sensors that are available to access and use to collect data. In addition to this, they are widely available with the majority of people owning one, which therefore does not limit the applications of my project. To access the sensors and record the data from the smartphone I decided on using a pre-existing app that is available on the Google play store called SensorRecord [1]. This app offers the ability to record data from a wide range of sensors available on the smartphone

[1] *https://play.google.com/store/apps/details?id=de.martingolpashin.sensor_record&hl=en_GB&gl=US&pli=1*

including accelerometer and gyroscope at any desired recording frequency. For each recording made, a folder is created in the phone's file system that contains a CSV file for each of the sensors selected and stores the timestamps, the milliseconds and the data values for the x, y and z axis of the sensor. An example of one of the sensor recording CSV files produced by the SensorRecord app is highlighted in Figure 3.



```
Timestamp,Milliseconds,X,Y,Z
2022-11-04 09:59:19,2,1.7357962,-9.498097,2.2337902
2022-11-04 09:59:19,102,1.9081788,-9.31135,2.2601264
2022-11-04 09:59:19,202,1.400608,-9.718364,2.2481554
2022-11-04 09:59:19,302,1.8124107,-9.325715,2.5115175
2022-11-04 09:59:19,402,3.9264908,-9.012074,2.717419
2022-11-04 09:59:20,503,0.260968,-9.227552,1.8124107
2022-11-04 09:59:20,604,2.753332,-7.0631943,-0.0047884034
2022-11-04 09:59:20,702,-3.2010477,-6.7208233,-4.8841715
2022-11-04 09:59:20,802,-7.5755534,-9.155726,-7.1038957
2022-11-04 09:59:20,902,-5.7727194,-7.3720465,-4.757279
2022-11-04 09:59:20,1002,-3.799598,-4.5801077,-7.580342
2022-11-04 09:59:20,1102,-6.9937625,-4.8506527,-6.4574614
2022-11-04 09:59:20,1203,-2.6910827,-4.0916905,-7.9442606
2022-11-04 09:59:20,1302,-3.25372,-6.004957,-6.380847
2022-11-04 09:59:20,1403,-4.1922474,-5.0353055,-7.2331824
2022-11-04 09:59:21,1503,-2.1978772,-6.2946553,-5.5835776
2022-11-04 09:59:21,1602,-3.383007,-7.5875244,-7.5420346
2022-11-04 09:59:21,1702,-2.693477,-6.6729393,-6.3257804
2022-11-04 09:59:21,1803,-4.000711,-6.7232175,-6.6609683
2022-11-04 09:59:21,1902,-3.430891,-6.830957,-6.4119716
2022-11-04 09:59:21,2003,-3.490746,-6.574777,-5.8517284
2022-11-04 09:59:21,2102,-4.7907977,-5.834969,-6.7734957
2022-11-04 09:59:21,2202,-3.2632968,-6.34254,-6.4239426
2022-11-04 09:59:21,2303,-4.99221,-6.728006,-6.356905
2022-11-04 09:59:21,2402,-3.9001546,-5.96665,-3.371036
2022-11-04 09:59:22,2502,-3.7253778,-8.20044,-6.3401456
2022-11-04 09:59:22,2603,-1.2641385,-9.5795,-6.6274495
2022-11-04 09:59:22,2702,-3.7181952,-10.494085,-6.289867
2022-11-04 09:59:22,2803,-3.3782187,-5.7128644,-6.6178727
2022-11-04 09:59:22,2902,-3.6655228,-5.789479,-6.1869164
2022-11-04 09:59:22,3004,-3.821146,-5.633856,-6.335357
2022-11-04 09:59:22,3102,-4.414908,-5.741595,-7.1493855
2022-11-04 09:59:22,3202,-6.5771713,-5.514146,-6.3616934
2022-11-04 09:59:22,3303,-3.1675289,-8.6697035,-6.182128
```

Figure 3: An example of the CSV file containing the raw sensor data produced by the SensorRecord app

For this project, a recording frequency of 10Hz was chosen because it provides a good balance between capturing enough data to be able to see the characteristics of the gym activities through the accelerometer and gyroscope sensor. While simultaneously generating small enough files to allow for easy data collection on any smartphone. Therefore, not limiting the devices that can collect data for this project but also use the real-time classification app. Notably, the average size of a CSV file for one of the sensor recordings was only 14Kb.

Table 1: Number of instances recorded for participants on each exercise

Activity	**Participant 1**	**Participant 2**
Bench Press	45	10
Squats	45	10
Deadlift	45	10
Military Press	45	10
Sit Ups	45	10
Press ups	45	10
Pull Ups	45	10
Stationary	45	10

In total, 440 data instances were recorded for the dataset of this project, a breakdown of the number of instances can be seen in Table 1. For this dataset, two participants were used to collect the data, with all the recordings being made with the same smartphone while attached to the upper left arm of the participant performing the exercises. This location was chosen because arm movement is required for all the activities, providing us with the most information about the movement of the exercise. Whereas a phone location such as a trouser pocket would not detect anywhere near the same amount of movement compared to the arm location for the majority of the exercises. Finally, for all exercises recorded a repetition range of around 6 to 12 was performed, generating 15-40 seconds of usable data per data instance.

3.2. Machine Learning Classifiers

This project utilised a range of different classifiers, including some common machine learning algorithms that are implemented in Python (created by Van Rossum and Drake (2009)) using the sci-kit learn tool-kit (developed by Pedregosa et al. (2011)), as well as some time series specific classifier algorithms that have been specifically developed to solve Time Series Classification problems, which have been implemented in Python using the sktime tool-kit (developed by Löning et al. (2019)). An overview of all the different classifiers used in the project is given below:

3.2.1. K Nearest Neighbours

The K Nearest Neighbours (kNN) classifier is a supervised machine learning algorithm that is used for classification and regression problems. This algorithm works by classifying a new data instance to the majority class of the k most similar data instances in the training set. For a TSC problem, a kNN classifier works essentially the same by calculating the distance between two time series and assigning the time series the class of the k most similar time series in the training set. To calculate the similarity between time series many different distance metrics can be used, for this project I will be using two different metrics which are Euclidean Distance (ED) and Dynamic Time Warping (DTW).

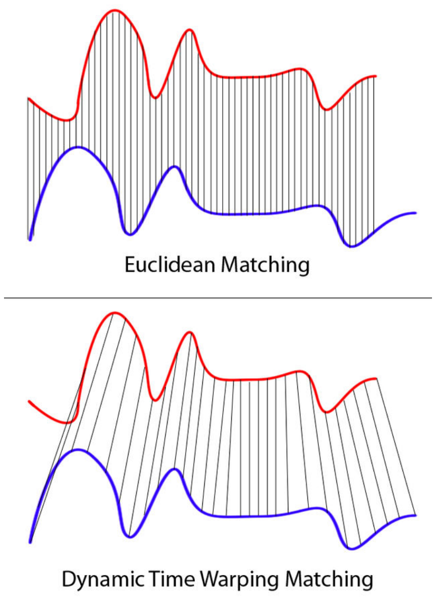


Figure 4: A diagram showing a comparison of how Euclidean Distance and Dynamic Time Warping are calculated

Euclidean Distance is the most used distance metric used for any common classification problem and calculates the distance between two points, which is shown below in Formula 1. ED can also be used for TSC problems and is calculated by getting the ED between linearly mapped data points in two time series. However, ED tends not to be appropriate for time series problems because it assumes that the two time series are aligned in time, which often is not the case with the majority of time series data. Since they usually have misalignments in time and are not always the same length, which ED can be sensitive to, resulting in poor performance from a classifier using this distance metric.

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{1}$$

Therefore, it is better to use a time series specific distance metric that can overcome these issues, such as DTW. DTW does this by allowing some warping between two time series to minimise the misalignments in the data and to find time series that are most similar, Lines and Bagnall (2015). A visual representation of how both distance metrics are calculated on two time series is shown in Figure 4.

3.2.2. Decision Tree

The Decision Tree algorithm is a popular supervised machine learning classifier that can be used for classification and regression problems. The algorithm works by recursively partitioning the training dataset into subsets based on decision rules learned from finding similarities in the data's attributes to create a decision tree model. As a result, a decision tree containing decision nodes and leaf nodes is formed. A data instance is then classified by traversing the tree and following the decision rules at each node until it reaches a leaf node where it is then classified.

3.2.3. AdaBoost

The AdaBoost algorithm, proposed by Freund and Schapire (1997), is a boosting ensemble method that works by training weak learning classifiers on repeatedly modified versions of the dataset. On the first iteration, each data instance in the dataset is initially given an equal weighting, giving all the data instances equal importance in the first model. After each iteration, the weights for the data instances are updated depending on if they were correctly classified or not. Instances that were miss-classified are given higher weights, so that they have more importance in the next iteration to hopefully increase the performance of the next model, by focusing on these instances more. The predictions of these classifiers are then combined through a majority vote to produce a final prediction.

3.2.4. Random Forest

Random Forest is a bagging ensemble method that works by training multiple Decision Trees on a random subset of features and data instances, which was proposed by Breiman (2001). Each random sample taken from the dataset uses replacement. The predictions from all the models are then combined to produce a final prediction for the ensemble, by using a major vote method to select the class that was predicted the most by all the models.

3.2.5. Naïve Bayes

Naïve Bayes is an algorithm that uses Bayes Theorem to find the probabilities of each class in the training data by using data features and the number of occurrences of each class in the training dataset. The algorithm assumes that all the features in the dataset are independent of each other when calculating the probabilities for each class. Naïve Bayes then makes predictions for new data instances by calculating the probabilities that a new data instance belongs to each class based on the features it is given and the previously calculated probabilities of each class. The class with the highest probability is then predicted as the class for that given data instance. The formula for Bayes Theorem is given below in Formula 2.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2}$$

3.2.6. BOSS

The BOSS (Bag of SFA Symbols) ensemble algorithm transforms time series data into a discrete set of symbols by using a sliding window across the time series and a technique known as SFA, which provides low pass filtering and quantisation to reduce noise in the time series, Schäfer (2015). The classifier then compares the time series based on the set of symbols produced for each series and looks for the time series with the most similar symbol representation.

3.2.7. Multi-Layer Perceptron

Multi-Layer Perceptron (MLP) is an artificial neural network which is made up of multiple hidden layers. Each layer contains many neurons, which are all connected to every neuron in the previous layer. The MLP algorithm works by initially setting the weights and biases of each neuron randomly. Training data is then passed into the first layer of the algorithm, which then produces outputs. These outputs are then used as the inputs for the following layer and a non-linear activation function such as Re-Lu is applied to the inputs. This process then continues until you reach the output layer which gives you a prediction. During the training phase, backpropagation happens after each prediction to minimise the difference between the network's prediction and the data instance's true label to optimise the performance of the network. This is done by adjusting the weights and biases in the network.

3.2.8. Time Series Forest

Time Series Forest (TSF) is a TSC specific ensemble algorithm, which constructs many Decision Tree classifiers through random sampling with replacement. The classifier makes a prediction for a time series data instance by using a majority vote method on all the predictions made from the Decision Tree models. Deng et al. (2013) explain that TSF uses a top-down, recursive strategy to build the Decision Trees but instead of using a standard splitting criterion such as entropy gain, uses a splitting criterion called entrance gain which is a combination of entropy gain and a distance measure.

3.2.9. Rocket

Rocket is a machine learning algorithm that is specifically developed for Time Series Classification. Rocket stands for Random Convolutional Kernel Transform. The algorithm that Dempster et al. (2020) has developed has a 'state of the art accuracy' which can compete and also beat many other time series algorithms for a fraction of the computational power they require. Rocket works by transforming time series data into a set of features using random convolutional kernels. The features that are acquired from this method are then used to train a linear classifier. Dempster et al. (2020) showed that it was effective to create multiple random convolutional kernels and when they are combined together, they can effectively find patterns in the time series data and find the features that are relevant for the classification of the time series.

3.3. Evaluation Metrics

To evaluate the performance of the classifiers used in this project, I have decided to calculate the following metrics based on conventional metrics typically used for TSC problems that were described by Dau et al. (2019) in Section 2.2. A description of all the metrics used is given below:

- **Accuracy** - The proportion of predictions that are correctly classified the same as the true label.
- **Balanced Accuracy** - an adaptation of accuracy that gives an average accuracy for each class in the dataset. This metric is typically used for datasets with class imbalances.
- **Train time** - the amount of time taken for a classifier to be trained on the training dataset. This metric is important for real-time classification because you will want to be able to choose a classifier that is able to be trained quickly to produce real-time predictions.
- **Precision** - refers to the percentage of predicted cases for a single class that was actually that class. For a multi-class dataset, the precision for the whole dataset can be calculated by taking an average of all the precisions for each class.
- **Recall** - is the percentage of cases for a particular class correctly classified. Similar to precision for a multi-class dataset, the recall for the whole dataset can be calculated by taking an average of all the recalls for each class.
- **F1 Score** - a metric that measures the overall accuracy of a model by considering the precision and recall of the model, which is calculated by taking the mean of the recall and precision.
- **AUROC** - Measures the area beneath the ROC curve, which is a plot of the true positive rate versus the false positive rate. AUROC is a useful evaluation metric since it indicates how good a model is at differentiating between classes. The higher the AUROC score, the more accurate the model is at classifying classes. AUROC is commonly used for binary classification problems, however, it can

also be adapted to be used for multi-class problems by using a one-versus-the-rest technique, which calculates the AUROC for each class against the rest of the classes and then calculates the average AUROC over all classes.

- **T-test** - a parametric statistical test used for determining whether or not there is a significant difference in the means of two groups. This will be used to determine whether there is a statistically significant difference in performance between a classifier and the other classifiers used.

3.4. Data Preprocessing

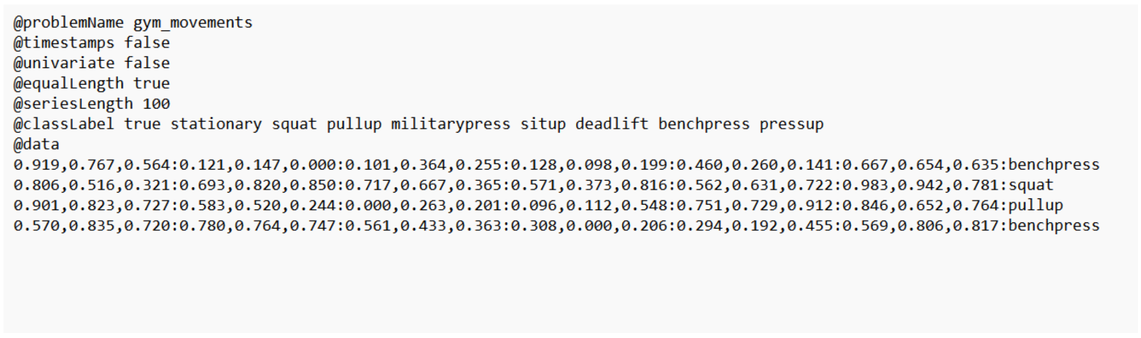
Before I could begin training the machine learning classifiers and conduct any experiments, I needed to preprocess the raw data I collected into a usable dataset that I could then use for training the sci-kit learn and sktime machine learning classifiers and perform analysis on. In this section, I will outline the steps I took to create the dataset.

The first step I took to create a dataset was to take all the individual data recordings generated by the SensorRecord app and combine the two CSV files generated by each recording for the accelerometer and gyroscope sensors into a single CSV file. Each combined CSV file then contained the timestamps, the milliseconds of when each sensor data was taken in the recording, the accelerometer data and the gyroscope data for each recording. To accomplish this, I wrote a custom Python script that iterated over every data instance recording directory and read both the CSV files for each recording into two Pandas DataFrames and merged them to create a single DataFrame that contained the columns mentioned earlier in the paragraph.

After this, I then removed the first 5 seconds of data from each of the weighted exercise recordings and the first 3 seconds of data for the body-weight recordings. The decision to remove the start of each recording was made to reduce noise that was captured from the setting up to record each exercise since the set-up of the movements was very random and did not represent the movement of the exercise. The reasoning behind removing different lengths of time for different exercises was made based on how long it took to set up to do an exercise and how long it took to perform each set for an exercise. Therefore, when looking at the data visually you could see that on average the set-up time for weighted exercises was longer at around 5 seconds, whereas for the bodyweight exercises the average set-up time was less, at around 3 seconds. This difference was caused due to having additional equipment to use for the weighted exercise, which in this case was a Barbell. Following this, I then extracted the first 10-second window of data remaining for each recording. The decision to use equal length recordings was made to keep the data consistent for all recordings across the dataset but also for easier implementation when training and testing the machine learning algorithms.

$$x_{normalised} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3}$$

Next, I normalised the data by scaling each instance into the range of 0-1 by using the min-max normalisation formula, which is highlighted in Formula 3. This was done to ensure that all the data was of a common scale and that only the underlying pattern of the exercise movement is captured and not how far off the ground the exercise is being performed. This was shown to influence the outcome of the models making the classification easier for the models which was discovered in my prototype experiment that is explained further in Section 3.6. Finally, I then saved the final 10-second Pandas DataFrame for each recording into a CSV file to create my formatted data instances.



```
@problemName gym_movements
@timestamps false
@univariate false
@equalLength true
@seriesLength 100
@classLabel true stationary squat pullup militarypress situp deadlift benchpress pressup
@data
0.919,0.767,0.564:0.121,0.147,0.000:0.101,0.364,0.255:0.128,0.098,0.199:0.460,0.260,0.141:0.667,0.654,0.635:benchpress
0.806,0.516,0.321:0.693,0.820,0.850:0.717,0.667,0.365:0.571,0.373,0.816:0.562,0.631,0.722:0.983,0.942,0.781:squat
0.901,0.823,0.727:0.583,0.520,0.244:0.000,0.263,0.201:0.096,0.112,0.548:0.751,0.729,0.912:0.846,0.652,0.764:pullup
0.570,0.835,0.720:0.780,0.764,0.747:0.561,0.433,0.363:0.308,0.000,0.206:0.294,0.192,0.455:0.569,0.806,0.817:benchpress
```

Figure 5: Example *ts* file that shows the data for 4 data recordings, each with 6 dimensions and 3 timepoints and a class label

Once I had my formatted data instances, I then had to transform all the individual CSV files into a file format that could store all the data as a single dataset and could be used by both sci-kit learn and sktime. For this, I used a bespoke time series file format created by sktime called a *ts file* that stores time series data in a standardised format that can be easily loaded and used in sktime. A *ts file* is made up of two parts: the header information and the data section. The header information section contains the metadata about the dataset including headers such as the problem name, if the data is univariate or multivariate and a list of all the possible class values. The data section follows this and begins after the @data tag, with each data instance being represented on an individual line, and each dimension for an instance being colon-delimited, with the final value being the data instance class label. Each value for a dimension is then represented as a comma-separated list, an example of a simple *ts file* problem can be seen in Figure 5.

The sktime library provides a range of utility functions that allow for creating and loading *ts files* into Python easily. Throughout this project, I used two of these utility functions, which were `write_ndarray_to_tsfile` and `load_from_tsfile`. To create my dataset I used the `write_ndarray_to_tsfile` utility function provided by sktime, this function works by taking in a 3D array of all the data instances in the format of (n_instances, n_columns, n_timepoints) and then a corresponding 1D array of the same length which contains the data instances class values. To convert the individual CSV files for each recording into the correct format required for the `write_ndarray_to_tsfile` function, I created a Python script that read all the data from the CSV files and appended the data from each one into a 3D array. Next, I extracted the class label for each data instance from the CSV filename and appended this to a 1D array. I made two variations of this script, the first created a single *ts file* containing all the data instances from the data I collected which was used in the 10-fold cross-validation experiment mentioned in Section 4.1.2. Whereas the second variation created two *ts files*, that used 50% of each classes data for each file to create a train and test split, which will be used in the single train-test split experiment which is also described in Section 4.1.2.

3.5. Dataset Variations

From all the data I collected three main dataset variations were created. The first was the main dataset called gym movements, which included the data for all 8 classes recorded. The next variation was the bodyweight movements dataset, which consisted of all the data instances for the sit-ups, press-ups and pull-ups classes. Finally, the last dataset variation was the weighted movements dataset, which only included the data of 4 classes: bench press, squats, deadlifts and military press. In total though, 15 different dataset variations were created from the 3 mentioned above, and each contained different sensor attributes to test on the classifiers to help answer my research questions outlined in Section 1. All dataset attributes used for each variation are shown in Table

2.

Table 2: Attributes used in each dataset variation

	Accelerometer			**Gyroscope**		
Dataset Name	**X axis**	**Y axis**	**Z axis**	**X axis**	**Y axis**	**Z axis**
Gym Movements	X	X	X	X	X	X
Gym Movements Accel	X	X	X	-	-	-
Gym Movements Gyro	-	-	-	X	X	X
Gym Movements ax	X	-	-	-	-	-
Gym Movements ay	-	X	-	-	-	-
Gym Movements az	-	-	X	-	-	-
Gym Movements gx	-	-	-	X	-	-
Gym Movements gy	-	-	-	-	X	-
Gym Movements gz	-	-	-	-	-	X
Bodyweight Movements	X	X	X	X	X	X
Bodyweight Movements Accel	X	X	X	-	-	-
Bodyweight Movements Gyro	-	-	-	X	X	X
Weighted Movements	X	X	X	X	X	X
Weighted Movements Accel	X	X	X	-	-	-
Weighted Movements Gyro	-	-	-	X	X	X

3.6. Prototype Experiment

In this project, I conducted three prototype experiments to test the viability of my data collection and pre-processing methodology, as well as whether it was possible to create a machine learning model that could successfully predict the gym exercise being performed. The same data collection and pre-processing methods described in Section's 3.1 and 3.4 were used for the prototype. However, data was only collected on a subset of the exercises utilised in the whole project and the first iteration of the prototype experiment used the dataset's real values and not a normalised dataset. The exercises used were squats, deadlifts, and bench press, resulting in a dataset of 56 data instances. All three prototype experiments used the same three classifiers: kNN with ED, kNN with DTW and Time Series Forest. All three of the classifiers were trained on 9 different variations of the dataset collected, each using a different selection of attributes which are outlined in Section 3.5.

In the first experiment, I used a single 70:30 train-test split which used the real values recorded from the sensor. Examining the results of this prototype experiment in the Appendix, Table 11 it can be seen that all the classifiers achieved 100% accuracy when using both accelerometer and gyroscope data together, as well as when using the multivariate accelerometer data and also when using only the *y axis* of the accelerometer. Therefore, this indicated to me that the *y axis* for the accelerometer data was having too much influence over the classifiers. This led me to discover that the ranges for each exercise's data were completely different depending on how high off the ground you were when you started the exercise and the range of motion of each exercise on a particular axis. This allowed the models to clearly distinguish the different exercises based on scale ranges. Therefore, making the classification too easy for the model.

As a result, I decided to normalise all of the data into a common scale between 0-1 for the next iteration of the prototype. The normalisation method used is described in Section 3.4. The results from the second prototype experiment showed kNN DTW still performed the best out of the three classifiers, achieving 100% accuracy on all of the multivariate dataset variations. Whereas, the accuracy for kNN ED dropped to 94.4% for both accelerometer and gyroscope data and then 83.3% for accelerometer and gyroscope data separately. TSF followed similarly with an accuracy of 94.4% for both the accelerometer and gyroscope together, 83.3% for just the accelerometer and 77.7% accuracy for just gyroscope data. In addition to this, the accuracy of just the *y axis* for accelerometer dropped from 100% for all the classifiers to 66.6% for kNN ED, 94.4% and then 83.3% for TSF. Showing the impact and influence that the different scale ranges were having on the classification results. All the results for the second prototype can be seen in the Appendix, Table 12.

Finally, in the last prototype experiment, I used a 50:50 train-test split with a normalised dataset. The results for this can be seen in the Appendix, Table 13 and show that the accuracy for kNN ED increased from 83.3% on the 70:30 split to 89.2% on the 50:50 split when using just the accelerometer data. However, the accuracy for just the gyroscope on the kNN ED classifier decreased from 83.3% on the 70:30 split to 78.6% on the 50:50 split. Examining the results of the 70:30 and 50:50 train-test splits results show that there is not any definite conclusion to say that more training data increases model performance, with some classifiers performing worse on more training data and others performing better on more training data. This is most likely caused due to the fact the dataset is very small for the prototype, meaning that small performance differences are exaggerated because of this.

Overall, the findings from my prototype experiment were that TSC can be used to predict gym exercises and that normalising the dataset ensures that there is no bias towards any particular axis due to different scales of values for an exercise based on how high off the ground you were when you started the exercise and the range of motion of each exercise on a particular axis. The results also show that out of the three classifiers used kNN DTW was the best classifier for this problem. With kNN DTW outperforming the other two classifiers on every dataset variation apart from one. Finally, the results also indicate that multivariate time series data produces better performing models for identifying gym movements in comparison to just univariate data.

4. Implementation and Evaluation

4.1. Experimental Plan

4.1.1. Experiments Performed

For this project, the main results that I was interested in were the ones that would answer my 5 research questions which are outlined in Section 1. To do this I ran five main experiments on the classifiers I described in Section 3.2 using the various dataset variation's that are given in Table 2. The five experiments that I performed and analysed the results of were:

- **8 Class gym exercise problem** - This experiment aims to answer my first research question: 'Can Time Series Classification algorithms be used to accurately classify between 8 different gym exercises?'. This experiment will be run on the full 8-class dataset of exercises collected by Participant 1 and will be trained and tested using both a 10-fold cross-validation method and 50:50 train-test split which are outlined below in Section 4.1.2. For this experiment, I hypothesised that all the time series specific classifiers i.e. Rocket, Boss, kNN DTW and TSF, would outperform all common machine learning algorithms used on the dataset, because they were designed specifically to find patterns and similarities between time series data, whereas the common algorithms are not.

- **Bodyweight or Weighted Exercises** - The aim of this experiment is to answer the research question 'Are weighted gym exercises or bodyweight exercises easier to classify using Time Series Classification algorithms?'. The experiment will contain two datasets: the first being a 3-class exercise dataset containing the following bodyweight exercises: pull-ups, sit-ups and press-ups and the second dataset consisting of the following weighted exercises: military press, deadlift, squat and bench press. All of the data used in both datasets will only contain the data from Participant 1 and will be trained and tested on all the classifiers specific in Section 3.2 using a 10-fold cross-validation method. I predict that the results of this experiment will show that neither bodyweight nor weighted exercises are easier to classify than the other because all of the exercises require you to move your body, providing distinguishable characteristics for each movement.

- **Multiple Sensors** - 'Does using both the accelerometer and gyroscope sensors together improve model performance over using just one of the sensors?' will be the research question explored in this experiment. All of the algorithms described in Section 3.2 will be trained and evaluated using a 10-fold cross-validation approach on the same exercise data instances used on the 8-class problem experiment. However, this will be done three times using three different data attributes, the first being just on the accelerometer data, the second purely gyroscope data, and finally both sensor's data combined. The results of all three runs will then be compared to determine which data feature combination creates the best-performing models. I hypothesise that using the combined data from both sensors will produce better results than using just the accelerometer or just the gyroscope data; combining data from both sensors will provide more information about an exercise's movement pattern.

- **Univariate or Multivariate** The research question: 'Is multivariate data better for classifying gym exercises than univariate data?' will be investigated in this experiment. This will be done using a 10-fold cross-validation on all the classifiers using variations of the 8-class exercise dataset from Participant 1, that include different sensor features to determine which type of data is best to use for this TSC problem. For this experiment, I expect the results to show that using multivariate data will produce better results than using univariate data because it takes into account multiple directions of acceleration and rotation on the phone compared to just one, which will help distinguish more clearly between different exercises.

- **Person Independent** - The person-independent experiment will look to answer the research question: 'Are the classifiers person dependent or will a model built on one person's data be able to accurately classify the gym exercises of another person?'. This experiment will be run on the dataset containing all 8 classes, with the training data being Participant 1's and the test data being Participant 2's. From this experiment, I predict that the results from the person-independent models will be worse due to physical differences between the participants, which will lead to different forms for each exercise.

4.1.2. Generating Data Splits

To evaluate the performance of the time series model's in the experiments outlined in Section 4.1.1, two approaches will be used to split my data into train and test splits. The first method is a 10-fold cross-validation, which is when the dataset is randomly split into 10 equally sized subsets, with nine of the subsets being used to train the model and the one remaining set being used to test the model performance. This process is then repeated 10 times, with each individual fold being used as the test data once. The evaluation metrics that I will calculate from the models are described in Section 3.3. The results obtained from all 10 folds will then be averaged out to give an overall performance for the model. This approach was chosen due to the overall size of my dataset being small. So, as recommended by Dau et al. (2019) in Section 2.2 that for time series problems with small data sets, re-sampling should be done multiple times to get the average performance metrics. The pseudocode for the 10-fold cross-validation approach I will be using on each classifier can be seen in the Appendix, Algorithm 1.

The second method I will be using to generate my train-test splits for my experiments is a single train-test split approach, the pseudocode for this can be seen in the Appendix, Algorithm 2. This approach will be conducted twice on two different dataset variations. The first run of this will be done using a 50:50 train-test split, using just the 8-class gym exercise data collected by Participant 1. This will be done to compare the results to the 10-fold cross-validation approach, to see if there is any significant difference in performance based on the size of the training data. However, it should be noted that due to the size of the dataset being small, differences in performance could be magnified as mentioned by Dau et al. (2019) in Section 2.2.

The second use of this method will be done to assess the models in the person-independent experiment. The training set for this method will consist of all the data collected by Participant 1, while the test data will consist of all the data collected by Participant 2. The same evaluation metrics used for the 10-fold cross-validation approach mentioned above will be used for the single train-test split datasets.

4.1.3. Experiment Implementation

The Python programming language was used to implement all of the experiments in this research, and the machine learning toolkits: sci-kit learn (developed by Pedregosa et al. (2011)) and sktime (developed by Löning et al. (2019)) were used to implement all of the algorithms. Sci-kit learn is an open-source machine learning package written in Python that includes a number of algorithms and tools for doing a variety of machine learning tasks such as classification, clustering, and regression. Sktime extends the sci-kit learn toolkit by providing a set of algorithms and tools that are designed for time series analysis.

For this project, all of the algorithms were implemented using their default parameters. The sci-kit learn toolkit was used to implement the following machine learning algorithms: kNN ED, Naïve Bayes, Decision Tree, MLP, AdaBoost, and Random Forest. All of these classifiers are common machine learning algorithms that were not designed for Time Series Classification; regardless, they can be used on time series data by ignoring the time relationship between the data features and by considering each data point as a distinct attribute when training the classifier. All of the remaining classifiers, including Rocket, TSF, Boss Ensemble, and kNN DTW, were implemented with sktime and support time series data. However, only TSF is unable to handle multivariate time series data natively, so an ensemble method provided by sktime called `ColumnEnsembleClassifier` was used to run this classifier on multivariate data. The `ColumnEnsembleClassifier` works by training a classifier on each dimension of the data and predicts using a majority vote method.

To perform all the experiments custom Python scripts were created. The scripts worked by iterating over all of the desired datasets for an experiment and loading the train and test data for each dataset. These were then used to train each classifier and make predictions on all of the test data, as well as calculate all of the performance metrics mentioned in Section 3.3. The results for each classifier were then saved in a *.txt file*, along with a confusion matrix image of all of the classifier's predictions. However, for the experiments that used a 10-fold cross-validation to train and test the classifiers, the sci-kit learn function called `StratifiedKFold` was used to create the folds on the whole dataset. A `StratifiedKFold` was used to make sure that each fold contained a proportional number of each class from the entire dataset. In addition, a seeded random state was set on the `StratifiedKFold` to ensure that all of the folds were reproducible and could be used on each classifier to ensure that the way the data was split did not effect classifier performance.

4.2. Analysis of Results

4.2.1. 8 Class gym exercise problem

The primary objective of this experiment was to determine which classifier performed best on the classification of the full gym exercise dataset, the results of the 10-fold cross-validation in this experiment are displayed in Table 3 and show the average performance metrics for each of the classifiers used across the 10 folds.

Table 3: Cross validation results for the gym movements dataset classification

Classifier	Accuracy	Balanced Accuracy	Precision	Recall	F1 Score	AUROC	Train time (m/s)
AdaBoost	0.192	0.193	0.155	0.192	0.129	0.581	0.881
Boss Ensemble	0.978	0.976	0.982	0.978	0.976	0.996	2323.285
Decision Tree	0.433	0.433	0.472	0.433	0.426	0.676	0.090
kNN-ED	0.858	0.857	0.882	0.858	0.856	0.973	0.000
kNN-DTW	0.978	0.976	0.982	0.978	0.977	0.987	0.005
Multi-layer Perceptron	0.706	0.709	0.723	0.706	0.701	0.932	0.819
Naïve Bayes	0.761	0.764	0.795	0.761	0.762	0.939	0.002
Random Forest	0.822	0.823	0.845	0.822	0.819	0.976	0.359
Rocket	0.992	0.991	0.993	0.992	0.991	0.995	1.231
Time Series Forest	0.942	0.941	0.953	0.942	0.941	0.993	23.204

The main findings that can be taken from Table 3 are that all the time series classifiers achieved better accuracy than all the common machine learning algorithms in the experiment. The worst performing time series algorithm was TSF, which achieved an accuracy of 94.2%, a result that is 8.4% higher than the best common machine learning algorithm, which was kNN ED which achieved an accuracy of 85.8%. This result was expected because the time series classifiers are designed to solve time series problems, whereas the traditional machine learning algorithms are not.

Rocket was the best-performing classifier on the dataset achieving an average accuracy of 99.2%, this was closely followed by Boss Ensemble and kNN DTW which both achieved an accuracy score of 97.8%. The most surprising outcome of this experiment was the performance of the AdaBoost classifier. AdaBoost was only able to achieve an accuracy of 19.2%, a result that is only 6.75% higher than the expected accuracy of the dataset if a random selection was used to predict the gym exercise being performed. This demonstrated that AdaBoost struggled immensely on the dataset and is not a good classifier to use for this time series problem.

Comparing the confusion matrices of both Rocket and Adaboost - which were the best and worst performing classifiers in the experiment, show that Rocket was able to correctly predict the time series data for 357 of the 360 data instances classified, with it only predicting 2 instances of military press and 1 stationary instance wrong. However, looking at the confusion matrix of AdaBoost, it can be seen that the classifier predicted 205 of the possible 360 data instances as either a deadlift or pull-up exercise, showing that the classifier could not distinguish the difference between the other exercises and these two exercises. The confusion matrices for both of these classifiers' predictions can be seen below in Figure's 6 and 7; the remaining confusion matrices of the classifiers used in this experiment can be found in the Appendix, Figures 9 - 16.

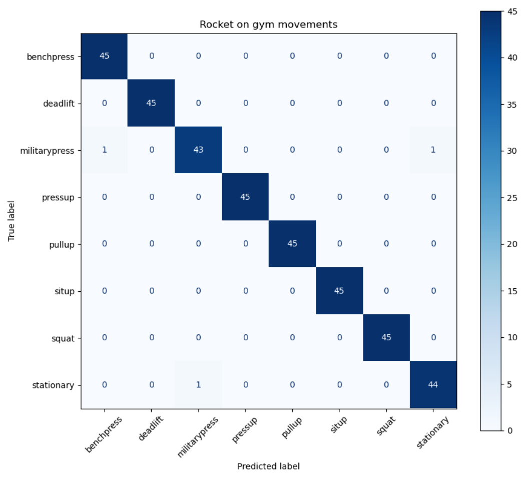


Figure 6: A confusion matrix showing the predictions of the Rocket classifier for each class in the gym dataset

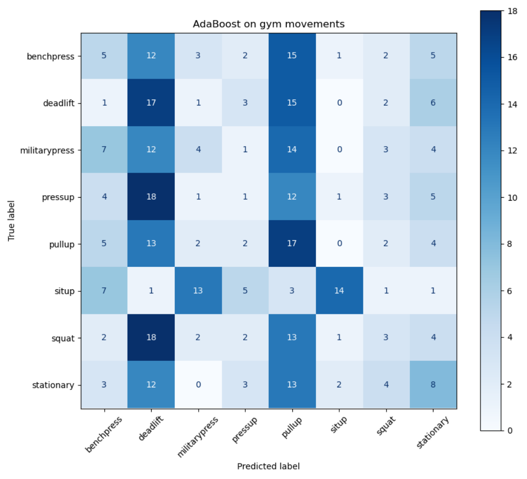


Figure 7: A confusion matrix showing the predictions of the AdaBoost classifier for each class in the gym dataset

Table 4: T-test showing if there is a significant difference in performance for all the classifiers against the Rocket Classifier

Classifier	**Mean Difference (%)**	**Mean Standard Deviation (%)**	**T-Value**	**Critical Region**
AdaBoost	80.00	7.38	34.26	2.26
Boss Enemble	1.39	2.36	1.86	2.26
Decision Tree	55.83	8.12	21.74	2.26
kNN ED	13.33	4.86	8.67	2.26
kNN DTW	1.39	1.96	2.24	2.26
Multi-layer Perceptron	28.61	5.72	15.83	2.26
Naïve Bayes	23.06	6.81	10.71	2.26
Random Forest	16.94	4.62	11.60	2.26
Time Series Forest	5.00	3.66	4.32	2.26

Because Rocket achieved the highest accuracy, a two-tailed t-test was conducted on the accuracy scores for each fold for Rocket versus every other classifier used. This was done to determine if Rocket performed significantly better than all the other classifiers and was the outright best classifier to use for this TSC problem. For Rocket to be considered significantly better than any other classifier, the T-value for the test must exceed the critical region, which in this case was 2.26 since a degree of freedom of 9 and a confidence level of 0.05 was used. The t-test results for this can be seen in Table

4. The outcome of the t-test performed determined that there is no significant difference in the performance of the Rocket, Boss Ensemble, and kNN DTW classifiers on this time series problem and therefore any three of the classifiers would be a good choice to use for a real-time classification app for accurately classifying gym exercises. However, the remaining 7 classifiers all exceeded the T-value of 2.26 and therefore are considered to perform significantly worse than the Rocket classifier on the 8-class gym exercise problem and are not in the same performance clique as the Rocket, KNN DTW and Boss Ensemble classifiers.

Table 5: Accuracy of different algorithms on the 8-class problem with 50:50 train-test data split

Classifier	**Accuracy**
AdaBoost	0.141
Boss Ensemble	0.967
Decision Tree	0.408
kNN ED	0.739
kNN DTW	0.978
Multi-layer Perceptron	0.609
Naïve Bayes	0.679
Random Forest	0.755
Rocket	0.989
Time Series Forest	0.875

The 8-class experiment was also run on a 50:50 train-test split to see if the amount of training data used impacted the performance of the classifiers. The results of the classifiers on this train-test split are highlighted in Figure 5. Comparing the results from the 50:50 train-test split to the results of the 10-fold cross-validation in Table 3 it can be seen that all the classifiers obtained higher accuracies when evaluated using cross-validation, except for kNN DTW which achieved the same accuracy on both train-test split methods. The average increase in accuracy when using the cross-validation was 6.21%, with the largest difference coming from kNN ED, which saw a difference of 11.90% increase in accuracy. Therefore, analysing this data suggests that using a larger training dataset increases the performance of the classifiers on this problem.

Overall, the outcomes of this experiment show that time series algorithms outperform common machine learning algorithms on this time series problem, allowing me to accept my hypothesis for this experiment. The experiment also showed that the best-performing algorithm on this problem was Rocket, which achieved the highest accuracy of 99.2%. However, it was in the same performance clique as kNN DTW and Boss Ensemble. Therefore, any three of the classifiers could be used for this problem. Finally, the results show that using more training data improves the classifier performance by 6.21% on average.

4.2.2. Bodyweight or Weighted Exercises

The results in Table 6 shows the mean balanced accuracy of each classifier over the 10 folds when evaluated on the bodyweight exercises and then on the weighted exercises. Table 6 also displays the difference of mean balanced accuracy between the bodyweight and weighted exercises for each classifier, with a positive value indicating that the classifiers had higher balanced accuracy on the bodyweight dataset and negative values showing that classifiers performed better on the weighted dataset. For this experiment balanced accuracy was analysed because of there being an unbalanced number of classes between the datasets, which balanced accuracy takes into account and is not influenced by, making it the best evaluation metric for this experiment.

Table 6: Balanced accuracy of different algorithms on bodyweight and weighted datasets

Classifier	**Bodyweight**	**Weighted**	**Difference**
AdaBoost	0.678	0.499	0.180
Boss Ensemble	1.000	0.989	0.011
Decision Tree	0.880	0.564	0.316
kNN ED	0.968	0.830	0.138
kNN DTW	1.000	0.976	0.024
Multi-layer Perceptron	0.895	0.831	0.064
Naïve Bayes	0.862	0.819	0.043
Random Forest	0.970	0.883	0.088
Rocket	1.000	1.000	0.000
Time Series Forest	0.993	0.936	0.057

Examining the results in Table 6 shows that all the classifiers performed better on the bodyweight exercise dataset, except for Rocket which achieved 100% accuracy on both datasets. The Decision Tree classifier saw the greatest difference in balanced accuracy between the two types of exercises, with a 31.6% difference in classifier performance between the two datasets, achieving a balanced accuracy of 88.0% on the bodyweight exercise dataset compared to only 56.4% on the weighted exercise dataset. Furthermore, AdaBoost saw a difference in balanced accuracy of 18.0% in favour of bodyweight exercises, with the bodyweight dataset obtaining a balanced accuracy of 67.8% and the weighted exercise dataset only getting a balanced accuracy of 49.9%.

These results imply that bodyweight exercises are easier to classify than weighted exercises, with the average difference between the balanced accuracy of the two types of exercise being 9.2% greater for bodyweight exercises. Additionally, the results show that common machine learning algorithm approaches seem to find it harder to classify weighted exercises correctly in comparison to the time series specific classifiers.

Table 7: T-test showing if there is a significant difference in classifying bodyweight and weighted exercises for all the classifiers

Classifier	**Mean Difference (%)**	**Mean Standard Deviation (%)**	**T-value**	**Critical Difference**
AdaBoost	17.96	27.54	2.06	2.26
Boss Ensemble	1.13	2.39	1.49	2.26
Decision Tree	31.89	14.69	6.87	2.26
kNN ED	13.83	9.19	4.76	2.26
kNN DTW	2.38	4.27	1.76	2.26
Multi-layer Perceptron	6.38	10.93	1.84	2.26
Naïve Bayes	4.29	11.14	1.22	2.26
Random Forest	8.75	7.65	3.62	2.26
Rocket	0.00	0.00	0.00	2.26
Time Series Forest	5.71	8.42	2.14	2.26

To see if there was a significant difference in how difficult the two types of exercises are to classify a t-test was performed on the balanced accuracy scores produced for the 10 folds for both datasets on all the classifiers, using a degree of freedom of 9 and a confidence level of 0.05. Table 7 shows the results of this t-test on the paired sample means, with the main finding being that only three of the ten classifiers used showed a significant difference in accuracy for identifying bodyweight exercises versus weighted exercises. The three classifiers with a significant difference were Decision Tree (T-value: 6.87), kNN ED (T-value: 4.76), and Random Forest (T-value: 3.62).

Overall, the findings of this experiment indicate that bodyweight exercises are easier to predict than weighted exercises with 9 out of the 10 of the classifiers obtaining higher balanced accuracies on the bodyweight exercises. Despite this, there does not appear to be a significant difference in how difficult the different exercise types are to classify on the different algorithms. Only 3 of the 10 classifiers showed a significant difference in performance. Based on this evidence, I cannot accept my hypothesis for this experiment because the difference is not statistically significant between the two types of exercise, despite some evidence suggesting that bodyweight exercises are easier to classify.

4.2.3. Multiple Sensors

Table 8: Accuracy results for each classifier trained on accelerometer data, gyroscope data and accelerometer and gyroscope data combined

	Accelerometer	**Gyroscope**	**Accelerometer and Gyroscope**
AdaBoost	0.178	0.236	0.192
Boss Ensemble	0.933	0.953	0.978
Decision Tree	0.436	0.472	0.433
kNN-ED	0.828	0.744	0.858
kNN-DTW	0.967	0.969	0.978
Multi-layer Perceptron	0.667	0.633	0.706
Naïve Bayes	0.661	0.608	0.761
Random Forest	0.761	0.783	0.822
Rocket	0.989	0.983	0.992
Time Series Forest	0.853	0.847	0.942

Table 8 shows the mean accuracy results for each classifier across the 10 folds when trained on just the accelerometer sensor, the gyroscope sensor and when trained on both of the sensor's data together. Looking at these results it is clear to see that using both the sensor's data together to train the classifier models produces better performing models than using just a single sensor, with 8 out of the 10 classifiers used in this experiment obtaining greater accuracies when trained on both the sensors data compared to using just one of the sensors. The classifiers that saw the biggest advantages from using both the sensors were Naïve Bayes, which saw a 15.28% increase over just using the gyroscope sensor and a 10% increase over using the accelerometer sensor, TSF which saw a 9.44% increase over just the gyroscope sensor and an 8.89% over just accelerometer and kNN ED that saw an 11.29% increase over gyroscope and just a 3.06% increase over just the accelerometer sensor. In contrast, the two classifiers that performed worse on the combined sensors were AdaBoost and Decision Tree, which both saw better accuracies when using just the gyroscope sensor. However, the differences were relatively small with AdaBoost seeing a 4.44% better accuracy for just the gyroscope sensor and Decision Tree having a 3.89% better accuracy when using just the gyroscope data.

Comparing both sensors individually to see if one is better than the other for collecting and predicting gym exercises it can be seen that half of the classifiers produced higher accuracies using accelerometer sensor data over gyroscope data and the other half produced better accuracies when using gyroscope data over accelerometer data. The classifiers that performed better on accelerometer data were: kNN ED, MLP, Naïve Bayes, Rocket and TSF. Whereas, the remaining classifiers performed better on gyroscope data. Anecdotally, this suggests that if only one of the sensors were used on this TSC problem, it would make no difference which sensor was used, with both sensors performing equally well on the same number of classifiers and the largest difference in performance on a single classifier being 8.33%.

In conclusion, the findings from this experiment show that using the combined data to train and evaluate the classifiers produced better-performing models than the use of just a single sensor's data, with 8 of the 10 classifiers producing more accurate models. Therefore, for this experiment, I can accept my hypothesis.

4.2.4. Univariate or Multivariate

Table 9: Accuracy results for each classifier on the univariate and multivariate dataset variations on the 8-class gym dataset

	Accel XYZ	**Accel X**	**Accel Y**	**Accel Z**	**Gyro XYZ**	**Gyro X**	**Gyro Y**	**Gyro Z**	**Accel & Gyro XYZ**
AdaBoost	0.178	0.272	0.175	0.161	0.236	0.194	0.142	0.242	0.192
Boss Ensemble	0.933	0.822	0.947	0.650	0.953	0.786	0.603	0.914	0.978
Decision Tree	0.436	0.397	0.400	0.256	0.472	0.281	0.158	0.497	0.433
kNN-ED	0.828	0.667	0.689	0.506	0.744	0.492	0.303	0.694	0.858
kNN-DTW	0.967	0.764	0.853	0.547	0.969	0.694	0.525	0.908	0.978
Multi-layer Perceptron	0.667	0.578	0.619	0.400	0.633	0.397	0.297	0.622	0.706
Naïve Bayes	0.661	0.547	0.536	0.369	0.608	0.414	0.306	0.558	0.761
Random Forest	0.761	0.636	0.675	0.453	0.783	0.522	0.358	0.744	0.822
Rocket	0.989	0.961	0.986	0.764	0.983	0.886	0.825	0.961	0.992
Time Series Forest	0.853	0.681	0.750	0.450	0.847	0.522	0.478	0.783	0.942

Examining the results in Table 9, which displays all of the classifier's performance on the univariate and multivariate dataset variations of the main full gym exercise dataset.

It can be observed that using multivariate accelerometer data provided more accurate models than using univariate accelerometer data for 8 of the 10 classifiers. The TSF algorithm was one of the algorithms that benefited significantly from utilising multivariate data over univariate data, achieving an accuracy of 85.3% on multivariate accelerometer compared to 68.1% on the *x axis*, 75.0% on the *y axis* and 45.0% on the *z axis*. The only two classifiers that produced better accuracy on univariate data were AdaBoost and Boss Ensemble. AdaBoost achieved the highest accuracy on the *x axis* with an accuracy of 27.2% compared to just 17.8% on the multivariate data. Whereas, Boss Ensemble obtained the highest accuracy on the *y axis* with an accuracy of 94.7% compared to 93.3% on multivariate.

Analysing the univariate and multivariate gyroscope data results reveals a similar trend to the accelerometer results, with 8 of the 10 classifiers performing best on the multivariate data, demonstrating that using multivariate data produces better performing models than just using univariate data. kNN DTW was one of the algorithms that saw gains from using multivariate gyroscope data over univariate data with an accuracy of 96.9% compared to 69.4% on the *x axis*, 52.5% on the *y axis* and 90.8% on the *z axis*. AdaBoost and Decision Tree were the only two classifiers that performed best on univariate data, with accuracies of 24.2% and 49.7% on the *z axis*, respectively. In comparison to their multivariate gyroscope accuracies of 23.6% and 77.2%.

Overall, the findings of this experiment imply that using multivariate data improves classifier performance over just using only univariate data, therefore I can accept my hypothesis for this research question.

4.2.5. Person Independent

Table 10: Accuracy of different algorithms on person-independent dataset

Classifier	Accuracy
AdaBoost	0.175
Boss Ensemble	0.350
Decision Tree	0.175
kNN ED	0.375
kNN DTW	0.775
Multi-layer Perceptron	0.288
Naïve Bayes	0.338
Random Forest	0.338
Rocket	0.563
Time Series Forest	0.313

To further test the classifier's on this TSC problem it was important to test the generalisation of the classifiers when trained on one person's data and tested on other people's data. This was done to determine if the models are person independent or if a training period would be required by users before they could use the machine learning model. Because there was only a small amount of test data available on the second participant for this experiment, a prototype was developed. Therefore, further experiments with more test data should be conducted in the future to further test the generalisation of the models and draw solid conclusions about the generalisation of this TSC problem.

All of the data recorded by Participant 1 was used for the training dataset, producing a training set of 360 instances, and all of the data recorded by Participant 2 was used for the test dataset, producing a total of 80 data recordings. Table 1 shows a breakdown of all the data instances recorded for each participant and class. The findings of this first experiment are shown in table 10.

Table 10 displays the results of this initial person-independent experiment. The main observation that can be made from these results is that the generalisation of this TSC problem on new unseen data from a second participant is poor with all the classifiers performing considerably worse in comparison to the results of the models produced from the experiment conducted in Section 4.2.1 that was trained and tested on one person's data.

The best-performing classifier on the person-independent dataset was kNN DTW achieving an accuracy of 77.5%. However, the most interesting result of this experiment is the performance of Boss Ensemble, which was one of the best-performing models on the person-dependent dataset (the dataset that consisted of just Participant 1's data). On the person-independent model, Boss Ensemble could only achieve an accuracy of 35.0%, compared to its accuracy of 97.8% on the person-dependent dataset. This is a significant difference in the performance of the Boss Ensemble classifier of 62.8%.

Looking at the confusion matrix of the Rocket classifier on this experiment in Figure 8 it can be seen that the model struggled to classify deadlift, military press, and pull-up movements between the two participants, with all the deadlift recordings being incorrectly classified as stationary movements, 7 of the 10 pull up movements being false positives for military press and 7 out of 10 military press being false positives for the bench press class. Therefore, this indicates that the form between the two participants for those exercises is different and looks more like other classes than their true class and that a training period for each user should be conducted to train a model before they use it. One possible reason for this could be because of the physical differences in the participants used, with Participant 1 being 6'4 and Participant 2 being smaller at 5'8. However, more data on a range of participants with varying physical characteristics should be tested to see if this has an impact on performance.

Overall, the results obtained from the experiment support my hypothesis, with all of the models performing worse compared to when they are trained and evaluated on a single participant's data, implying that the models have low generalisation on new unseen data recorded from different people. As a result, a training period should be implemented before using the machine learning models on new people. However, it is essential to acknowledge that these findings are preliminary and that further testing on a larger range of data is required to validate the results of this experiment before any definitive conclusions can be reached.

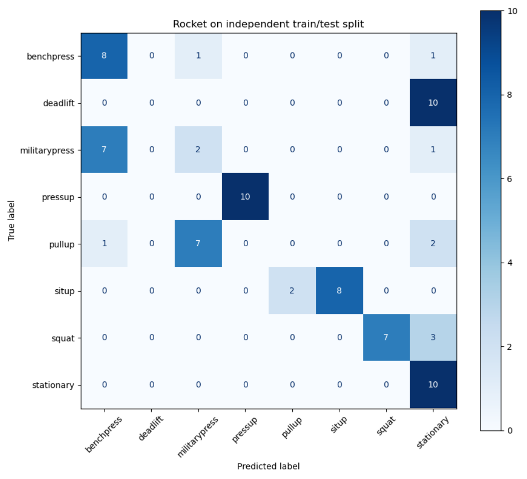


Figure 8: A confusion matrix showing the predictions of the Rocket classifier when trained on Participant 1's data and tested on Participant 2's data

4.3. Real-time exercise classification

An extension goal of this project was to create a real-time gym classification app to encourage and gamify exercise. From the findings discovered in my experiments and the time series models produced, I have developed a prototype Android app that classifies the gym exercise a user is performing in real-time based on the accelerometer and gyroscope readings recorded from the phone. The prototype app requires a user to attach the smartphone device to their upper arm when recording themselves doing exercise. The app can currently only predict the 8 exercises used in the TSC problem conducted in Section 4.2.1.

The application works by recording the accelerometer and gyroscope data for a 15-second period when the start button is pressed, capturing the user's phone movement during this time. The sensor data is then converted into a 2d JSON array and is sent as a post request to the backend server via the Volley Library. This JSON array is then converted into a numpy array, and the data is processed using the same method conducted to create the dataset in this project's machine learning aspect, which is explained in Section 3.4. The 10 seconds of sensor data is then passed into the machine learning model, which predicts the exercise. Example screenshots of the server doing this can be seen in the Appendix, Figures 20 and 21. This prediction is then returned as a response to the mobile app and displayed on the screen for the user to see, a screenshot of this can be seen in the Appendix, Figure 18.

To create this prototype, I used the Java Android SDK for the front-end which accesses and records the sensor data from the smartphone device and then sends this data to a backend server that has the serialised machine learning model running on it. For the backend server, a Python Flask server was used to deploy my machine learning model so that I could create real-time predictions. The model used for this prototype mobile app was the Rocket model which achieved 99.2% accuracy on the 8-class problem experiment in Section 4.2.1. This model was selected over Boss Ensemble even though there was no significant difference in performance between the two models because it achieved the highest overall accuracy and the train time of Rocket compared to Boss Ensemble was considerably better. Although kNN DTW could have also been used for this prototype with it having a faster train time than Rocket and there being no significant difference between the two models' performance. However, Rocket was ultimately decided due to it having the best accuracy overall and a fast training time of just 1.2 seconds.

5. Conclusion and Future Work

5.1. Conclusion

In conclusion, this project has demonstrated that Time Series Classification can be used to accurately predict gym exercises using accelerometer and gyroscope sensor data captured from a smartphone. This project achieved an accuracy of 99.2% on the problem and therefore achieved the project's aim. This project has also shown that Time Series Classification can be used to create an application which is capable of predicting gym exercises in real time.

The main findings from this project were that the Rocket algorithm was the best performing model in this project, achieving an accuracy of 99.2%. However, both the kNN DTW and Boss Ensemble algorithms were found to also be in the same performance clique as Rocket. Therefore, both are good alternative options for this problem, with both obtaining an accuracy of 97.7%. In addition to this, the results showed that combining accelerometer and gyroscope data provided better performing models than using either sensor's data individually and that using multivariate data improved algorithm performance over using only univariate data. The results also highlight that there is no significant difference in the difficulty of classifying bodyweight exercises compared to weighted exercises on the classifiers used. Finally, the results show that when using data from new people, the generalisation of the models on this TSC problem appears to be fairly poor compared to the generalisation of just a single participant. However, this was a shortcoming of this project due to the limited amount of data collected from other participants, which was caused by how time consuming it was to collect data for multiple people. This is definitely an improvement that can be made for this project by collecting more data on a wider range of people to further investigate this.

5.2. Future Work

To expand on the results of this project, there are many directions this project could be taken if given more time and resources. The first direction this project could be taken would be to collect data from a wider range of participants from varying fitness levels, abilities and physical characteristics to further investigate the generalisation of the classifiers and help improve performance. Another future direction of this project would be to develop a machine-learning model that is capable of detecting and counting individual repetitions of an exercise. To achieve this, I would look at collecting a new dataset that was labelled with the exercise performed as well as the number of repetitions performed. From there, I could then use supervised time series machine-learning classifiers to create models that are capable of accurately detecting and counting repetitions of various exercises. The final avenue I would like to expand this project is to further develop the real-time gym classification prototype app that I made. The features I would like to incorporate in my app would be to be able to track users' exercises in real-time to analyse their form and provide suggestions on how to improve. In addition to including a gamification aspect such as having achievements and goal settings to track users' progress in order to motivate and encourage them to exercise more.

References

Bagnall, A., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. (2018). The UEA multivariate time series classification archive, 2018. arXiv:1811.00075 [cs, stat].

Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1):5–32.

Dau, H. A., Bagnall, A., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., and Keogh, E. (2019). The UCR Time Series Archive. arXiv:1810.07758 [cs, stat].

Dempster, A., Petitjean, F., and Webb, G. I. (2020). ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495.

Deng, H., Runger, G., Tuv, E., and Vladimir, M. (2013). A time series forest for classification and feature extraction. *Information Sciences*, 239:142–153.

Freund, Y. and Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Khan, U. A., Khan, I. A., Din, A., Jadoon, W., Jadoon, R. N., Khan, M. A., Khan, F. G., and Khan, A. N. (2020). Towards a Complete Set of Gym Exercises Detection Using Smartphone Sensors. *Scientific Programming*, 2020:e6471438. Publisher: Hindawi.

Lines, J. and Bagnall, A. (2015). Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29(3):565–592.

Lines, J., Davis, L. M., Hills, J., and Bagnall, A. (2012). A shapelet transform for time series classification. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 289–297, Beijing China. ACM.

Löning, M., Bagnall, A., Ganesh, S., Kazakov, V., Lines, J., and Király, F. J. (2019). sktime: A Unified Interface for Machine Learning with Time Series. arXiv:1909.07872 [cs, stat].

MathWorks (2023a). Measure linear acceleration along X, Y, and Z axes in m/s2 - Simulink.

MathWorks (2023b). Measure rotational speed around X, Y, and Z axes in rad/s - Simulink.

Nurwanto, F., Ardiyanto, I., and Wibirama, S. (2016). Light sport exercise detection based on smartwatch and smartphone using k-Nearest Neighbor and Dynamic Time Warping algorithm. In *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 1–5.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Ruiz, A. P., Flynn, M., Large, J., Middlehurst, M., and Bagnall, A. (2021). The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2):401–449.

Schäfer, P. (2015). The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530.

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Wang, Y. C., McPherson, K., Marsh, T., Gortmaker, S. L., and Brown, M. (2011). Health and economic burden of the projected obesity trends in the USA and the UK. *The Lancet*, 378(9793):815–825.

A. Appendix

Algorithm 1 TimeSeriesExperimentCV(*dataset*,*clf*) **return** *avgMetrics*

```
Require: $dataset$, the full dataset file and the $clf$, the classifier to evaluate in the experiment
Ensure: $avgMetrics$, the average performance metrics of the classifier
 1: $data \leftarrow loadData(dataset)$
 2: $k \leftarrow 10$                                              ▷ set number of folds
 3: $evaluationMetrics \leftarrow []$          ▷ list to store evaluation metrics for each fold
 4: for $i \leftarrow 1 \in k$ do
 5:     $predictions \leftarrow []$                ▷ list to store predictions for each fold
 6:     $classLabels \leftarrow []$           ▷ list to store all the data instances true class
 7:     $trainData \leftarrow getTrainData(data, i, k)$      ▷ get training data for current fold
 8:     $testData \leftarrow getTestData(data, i, k)$           ▷ get test data for current fold
 9:     $classifier \leftarrow buildClassifier(clf)$
10:     $classifier.train(trainData)$          ▷ train the classifier on the training data
11:     for each $dataInstance, classLabel \in testData$ do
12:         $prediction \leftarrow classifier.predict(dataInstance)$
13:         $predictions \leftarrow predictions + prediction$
14:         $classLabels \leftarrow classLabels + classLabel$
15:     $metricResults \leftarrow CalculateMetrics(predictions, classLabels)$
16:     $evaluationMetrics \leftarrow evaluationMetrics + [metricResults]$
17: $avgMetrics \leftarrow average(evaluationMetrics)$    ▷ calculate average metrics over 10 folds
18: return $avgMetrics$
```

Algorithm 2 TimeSeriesExperiment(*trainDataset*,*testDataset*,*clf*) **return** *metrics*

```
Require: trainDataset, the train dataset file, testDataset, the test dataset file and the
    clf, the classifier to evaluate in the experiment.
Ensure: metrics, the performance metrics produced by the classifier on the test data
 1: $trainData \leftarrow loadData(trainDataset)$
 2: $testData \leftarrow loadData(testDataset)$
 3: $predictions \leftarrow []$                                    ▷ list to store predictions
 4: $classLabels \leftarrow []$                  ▷ list to store all the data instances true class
 5: $classifier \leftarrow buildClassifier(clf)$
 6: $classifier.train(trainData)$                   ▷ train the classifier on the training data
 7: for each $dataInstance, classLabel \in testData$ do
 8:     $prediction \leftarrow classifier.predict(dataInstance)$
 9:     $predictions \leftarrow predictions + prediction$
10:     $classLabels \leftarrow classLabels + classLabel$
11: $metricResults \leftarrow CalculateMetrics(predictions, classLabels)$
12: return $metrics$
```

Dataset	**kNN ED (%)**	**kNN DTW (%)**	**TSF (%)**
Accelerometer and Gyroscope	100.0	100.0	100.0
Accelerometer	100.0	100.0	100.0
Gyroscope	55.5	94.4	100.0
Accelerometer X-axis	44.4	77.7	83.3
Accelerometer Y-axis	100.0	100.0	100.0
Accelerometer Z-axis	77.7	77.7	83.3
Gyroscope X-axis	61.1	88.8	72.2
Gyroscope Y-axis	38.8	66.6	77.7
Gyroscope Z-axis	61.1	94.4	94.4

Table 11: Prototype 1 results, showing the accuracy of each classifier on the 70:30 train-test split real valued dataset

Dataset	kNN ED (%)	kNN DTW (%)	TSF (%)
Accelerometer and Gyroscope	94.4	100.0	94.4
Accelerometer	83.3	100.0	83.3
Gyroscope	83.3	100.0	77.7
Accelerometer X-axis	72.2	72.2	66.6
Accelerometer Y-axis	66.6	94.4	83.3
Accelerometer Z-axis	38.8	66.6	50.0
Gyroscope X-axis	66.6	94.4	66.6
Gyroscope Y-axis	50.0	72.2	61.1
Gyroscope Z-axis	83.3	94.4	83.3

Table 12: Prototype 2 results, showing the accuracy of each classifier on a 70:30 train-test split on a normalised dataset

Dataset	kNN ED (%)	kNN DTW (%)	TSF (%)
Accelerometer and Gyroscope	92.3	100.0	92.3
Accelerometer	89.2	100.0	85.6
Gyroscope	78.6	100.0	82.1
Accelerometer X-axis	75.0	67.9	60.7
Accelerometer Y-axis	67.9	89.3	78.6
Accelerometer Z-axis	46.4	67.9	50.0
Gyroscope X-axis	67.9	96.4	64.3
Gyroscope Y-axis	42.9	53.6	50.0
Gyroscope Z-axis	85.6	96.4	85.7

Table 13: Prototype 3 results, showing the accuracy of each classifier on a 50:50 train-test split on a normalised dataset

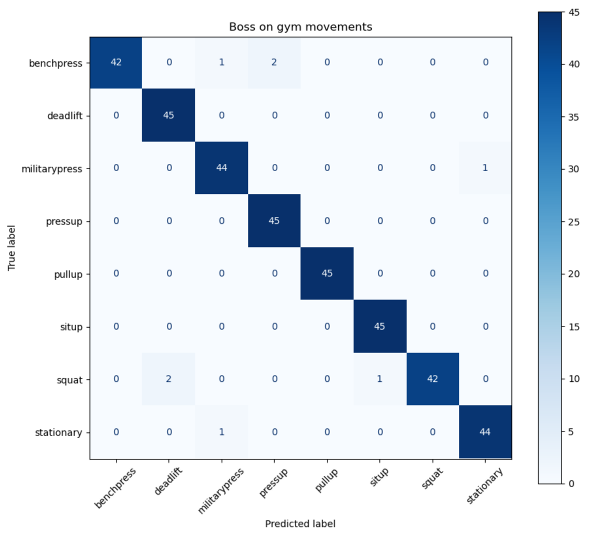


Figure 9: A confusion matrix showing the predictions of the Boss classifier for each class in the gym dataset

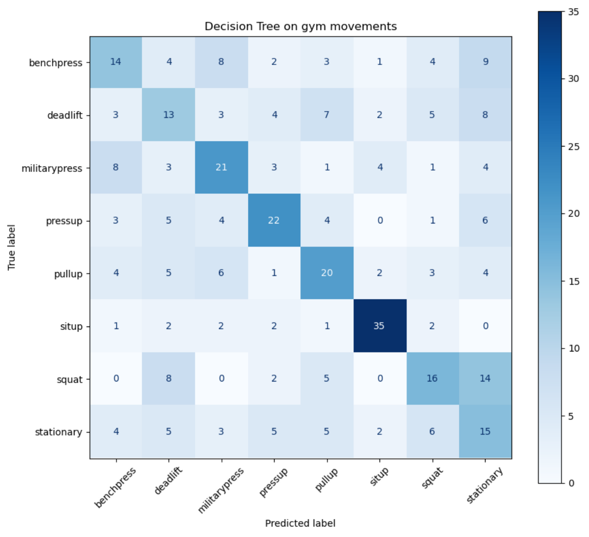


Figure 10: A confusion matrix showing the predictions of the Decision Tree classifier for each class in the gym dataset

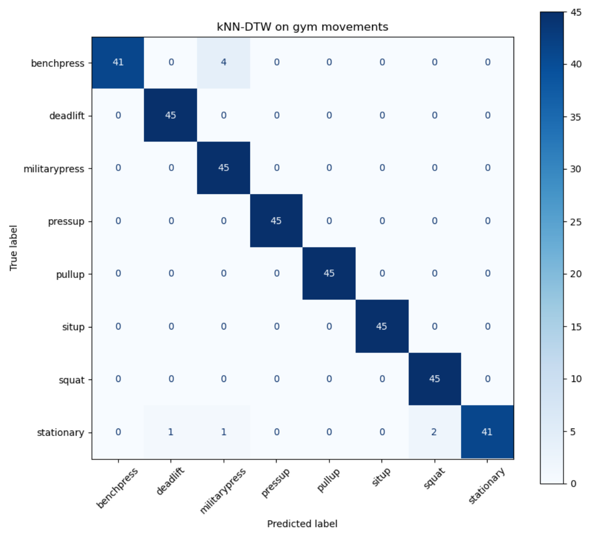


Figure 11: A confusion matrix showing the predictions of the kNN DTW classifier for each class in the gym dataset

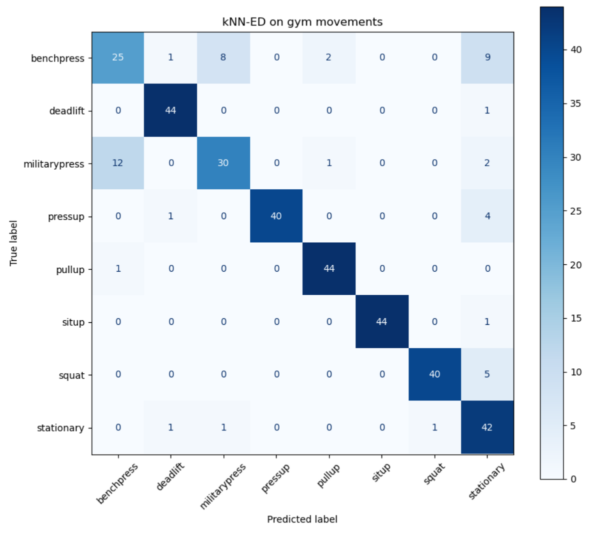


Figure 12: A confusion matrix showing the predictions of the kNN ED classifier for each class in the gym dataset

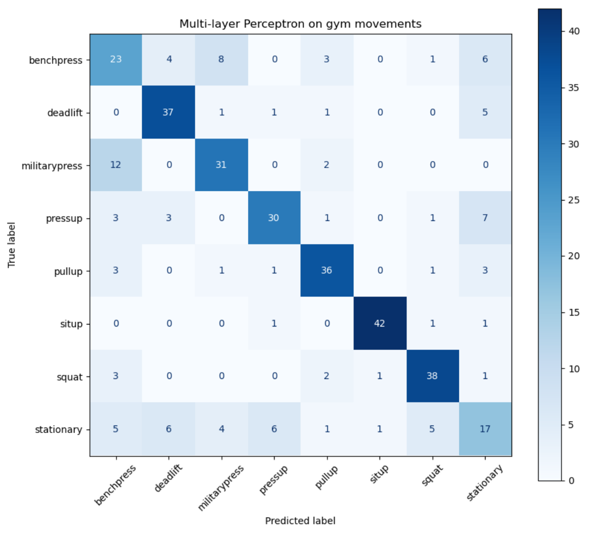


Figure 13: A confusion matrix showing the predictions of the MLP classifier for each class in the gym dataset

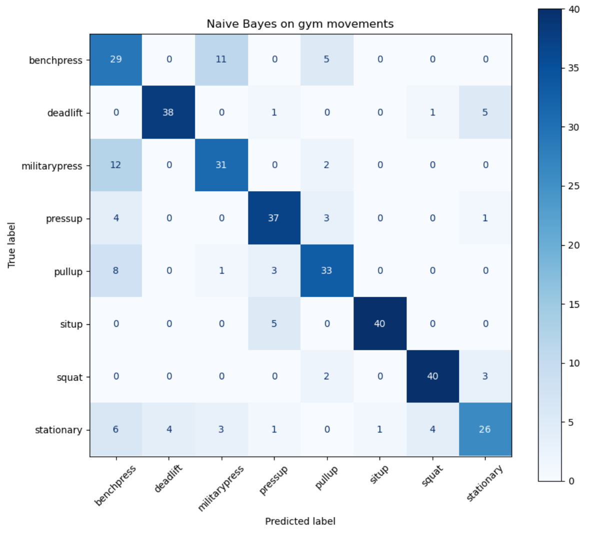


Figure 14: A confusion matrix showing the predictions of the Naïve Bayes classifier for each class in the gym dataset

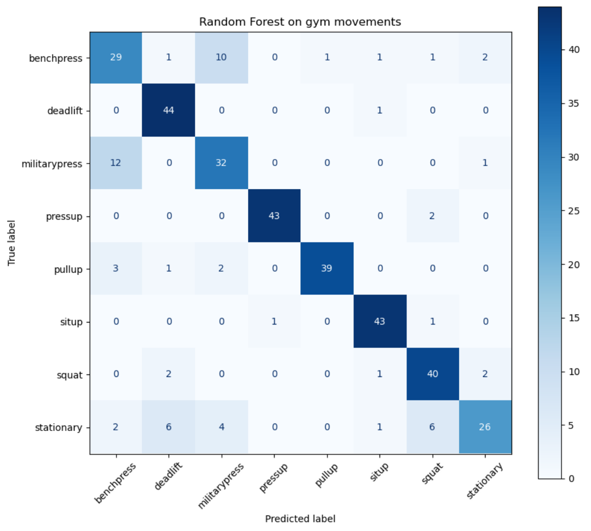


Figure 15: A confusion matrix showing the predictions of the Random Forest classifier for each class in the gym dataset

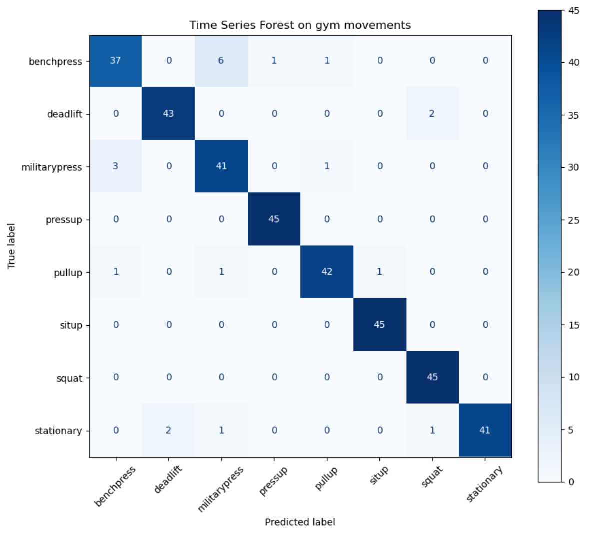


Figure 16: A confusion matrix showing the predictions of the TSF classifier for each class in the gym dataset

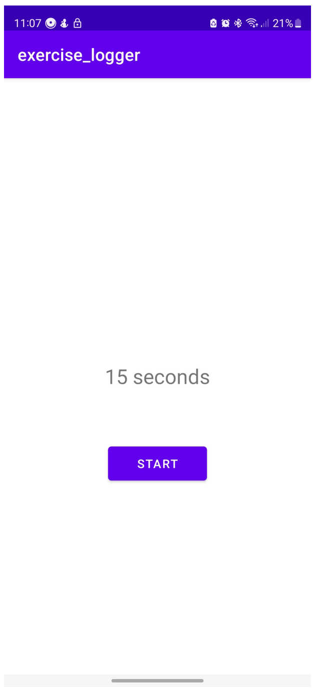


Figure 17: A screenshot of the prototype real-time classification app start screen

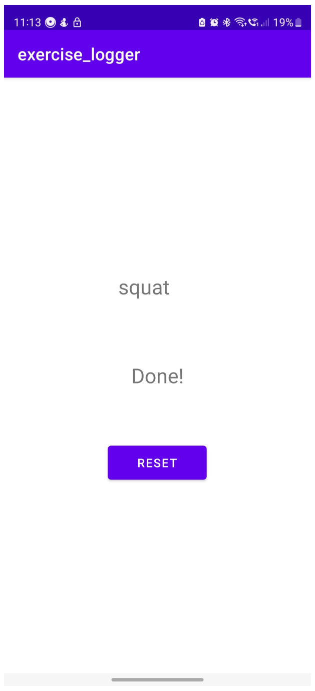
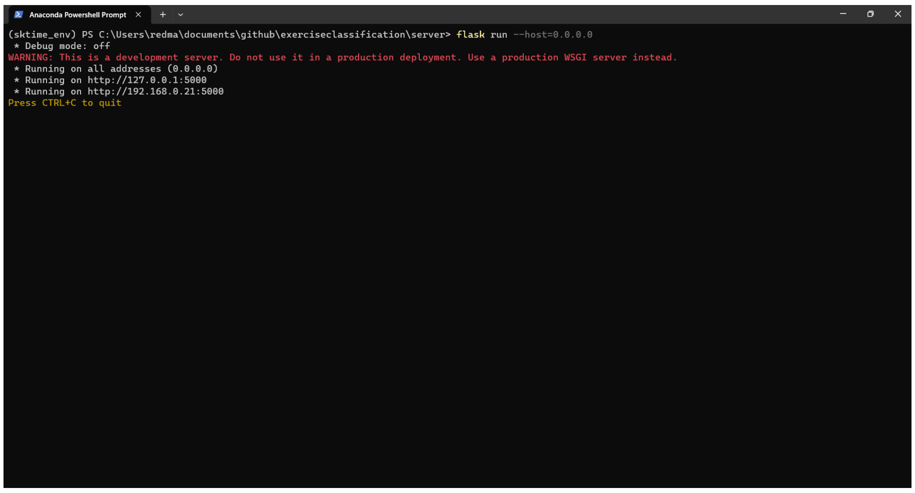
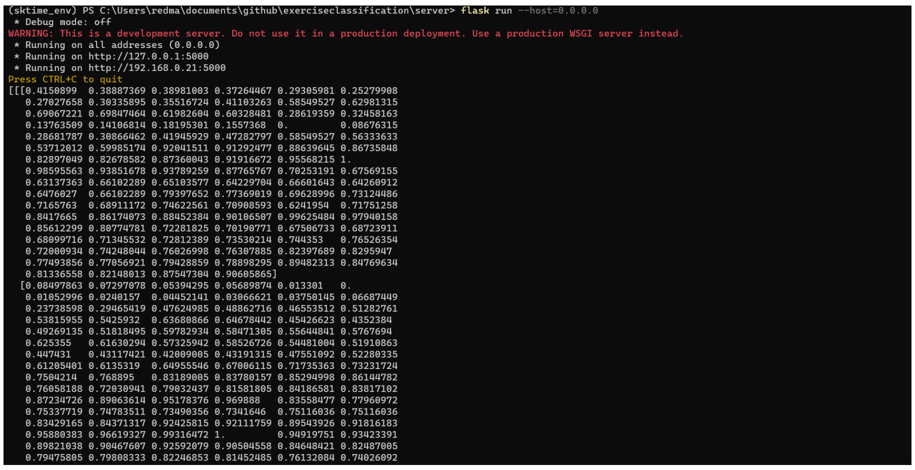


Figure 18: A screenshot of the prototype real-time classification app with the prediction made



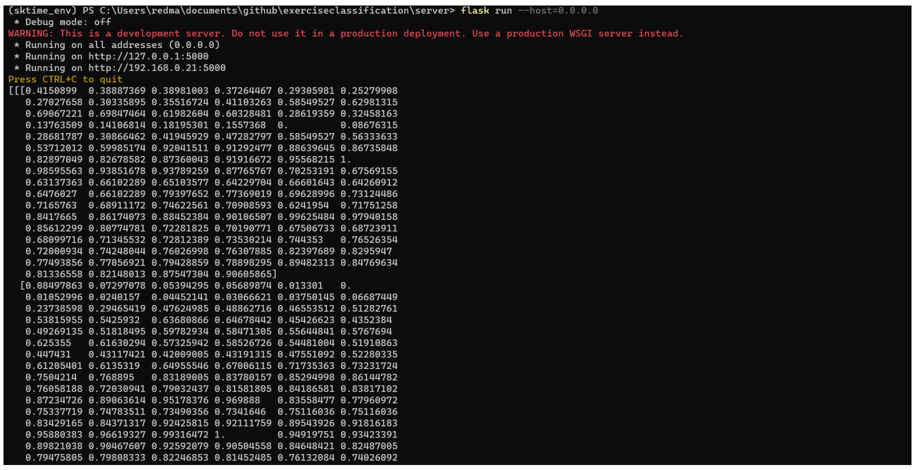
```
(sktime_env) PS C:\Users\redma\documents\github\exerciseclassification\server> flask run --host=0.0.0.0
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.0.21:5000
Press CTRL+C to quit
```

Figure 19: A screenshot of the development server running for the real-time classification app



```
(sktime_env) PS C:\Users\redma\documents\github\exerciseclassification\server> flask run --host=0.0.0.0
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.0.21:5000
Press CTRL+C to quit
[[[0.4150899  0.38887369 0.38981003 0.37264467 0.29305981 0.25279908
   0.27027658 0.30335895 0.35516724 0.41103263 0.58549527 0.62981315
   0.69067221 0.69847464 0.61982604 0.60328481 0.28619359 0.32458163
   0.13763509 0.14106814 0.18195301 0.1557368  0.         0.08676315
   0.28681787 0.30866462 0.41945929 0.47282797 0.58549527 0.56333633
   0.53712012 0.59985174 0.92041511 0.91292477 0.88639645 0.86735848
   0.82897049 0.82678582 0.87360043 0.91916672 0.95568215 1.
   0.98595563 0.93851678 0.93789259 0.87765767 0.70253191 0.67569155
   0.63137363 0.66102289 0.65103577 0.64229704 0.66601643 0.64260912
   0.6476027  0.66102289 0.79397652 0.77369019 0.69628996 0.73124486
   0.7165763  0.68911172 0.74622561 0.70908593 0.6241954  0.71751258
   0.8417665  0.86174073 0.88452384 0.90106507 0.99625484 0.97940158
   0.85612299 0.80774781 0.72281825 0.70190771 0.67506733 0.68723911
   0.68099716 0.71345532 0.72812389 0.73530214 0.744353   0.76526354
   0.72000934 0.74248044 0.76026998 0.76307885 0.82397689 0.8295947
   0.77493856 0.77056921 0.79428859 0.78898295 0.89482313 0.84769634
   0.81336558 0.82148013 0.87547304 0.90605865]
  [0.08497863 0.07297078 0.05394295 0.05689874 0.013301   0.
   0.01052996 0.0240157  0.04452141 0.03066621 0.03750145 0.06687449
   0.23738598 0.29465419 0.47624985 0.48862716 0.46553512 0.51282761
   0.53815955 0.5425932  0.63680866 0.64678442 0.45426623 0.4352384
   0.49269135 0.51818495 0.59782934 0.58471305 0.55644841 0.5767694
   0.625355   0.61630294 0.57325942 0.58526726 0.54481004 0.51910863
   0.447431   0.43117421 0.42009005 0.43191315 0.47551092 0.52280335
   0.61205401 0.6135319  0.64955546 0.67006115 0.71735363 0.73231724
   0.7504214  0.768895   0.83189005 0.83780157 0.85294998 0.86144782
   0.76058188 0.72030941 0.79032437 0.81581805 0.84186581 0.83817102
   0.87234726 0.89063614 0.95178376 0.969888   0.83558477 0.77960972
   0.75337719 0.74783511 0.73490356 0.7341646  0.75116036 0.75116036
   0.83429165 0.84371317 0.92425815 0.92111759 0.89543926 0.91816183
   0.95880383 0.96619327 0.99316472 1.         0.94919751 0.93423391
   0.89821038 0.90467607 0.92592079 0.90504558 0.84648421 0.82487005
   0.79475805 0.79808333 0.82246853 0.81452485 0.76132084 0.74026092
```

Figure 20: A screenshot of the development server receiving sensor data from the prototype Android app in real time



```
(sktime_env) PS C:\Users\redma\documents\github\exerciseclassification\server> flask run --host=0.0.0.0
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.0.21:5000
Press CTRL+C to quit
[[[0.4150899  0.38887369 0.38981003 0.37264467 0.29305981 0.25279908
   0.27027658 0.30335895 0.35516724 0.41103263 0.58549527 0.62981315
   0.69067221 0.69847464 0.61982604 0.60328481 0.28619359 0.32458163
   0.13763509 0.14106814 0.18195301 0.1557368  0.         0.08676315
   0.28681787 0.30866462 0.41945929 0.47282797 0.58549527 0.56333633
   0.53712012 0.59985174 0.92041511 0.91292477 0.88639645 0.86735848
   0.82897049 0.82678582 0.87360043 0.91916672 0.95568215 1.
   0.98595563 0.93851678 0.93789259 0.87765767 0.70253191 0.67569155
   0.63137363 0.66102289 0.65103577 0.64229704 0.66601643 0.64260912
   0.6476027  0.66102289 0.79397652 0.77369019 0.69628996 0.73124486
   0.7165763  0.68911172 0.74622561 0.70908593 0.6241954  0.71751258
   0.8417665  0.86174073 0.88452384 0.90106507 0.99625484 0.97940158
   0.85612299 0.80774781 0.72281825 0.70190771 0.67506733 0.68723911
   0.68099716 0.71345532 0.72812389 0.73530214 0.744353   0.76526354
   0.72000934 0.74248044 0.76026998 0.76307885 0.82397689 0.8295947
   0.77493856 0.77056921 0.79428859 0.78898295 0.89482313 0.84769634
   0.81336558 0.82148013 0.87547304 0.90605865]
  [0.08497863 0.07297078 0.05394295 0.05689874 0.013301   0.
   0.01052996 0.0240157  0.04452141 0.03066621 0.03750145 0.06687449
   0.23738598 0.29465419 0.47624985 0.48862716 0.46553512 0.51282761
   0.53815955 0.5425932  0.63680866 0.64678442 0.45426623 0.4352384
   0.49269135 0.51818495 0.59782934 0.58471305 0.55644841 0.5767694
   0.625355   0.61630294 0.57325942 0.58526726 0.54481004 0.51910863
   0.447431   0.43117421 0.42009005 0.43191315 0.47551092 0.52280335
   0.61205401 0.6135319  0.64955546 0.67006115 0.71735363 0.73231724
   0.7504214  0.768895   0.83189005 0.83780157 0.85294998 0.86144782
   0.76058188 0.72030941 0.79032437 0.81581805 0.84186581 0.83817102
   0.87234726 0.89063614 0.95178376 0.969888   0.83558477 0.77960972
   0.75337719 0.74783511 0.73490356 0.7341646  0.75116036 0.75116036
   0.83429165 0.84371317 0.92425815 0.92111759 0.89543926 0.91816183
   0.95880383 0.96619327 0.99316472 1.         0.94919751 0.93423391
   0.89821038 0.90467607 0.92592079 0.90504558 0.84648421 0.82487005
   0.79475805 0.79808333 0.82246853 0.81452485 0.76132084 0.74026092
```

Figure 21: A screenshot of the development server making a prediction based on the data it received from the app using the Rocket model