Data Mining Report

Bremen Big Data Challenge 2019
Human activity classification

by
Atit Bashyal
&
Rana Hamza Intisar

A report presented for the degree of
Masters of Science in Data Engineering

Prof. Dr. Adalbert F. X. Wilhelm
Jacobs University Bremen

Contents

1 Executive Summary

1.1 Task Outline and the Dataset

The Bremen Big Data Challenge 2019 was to classify various human movements which included everyday movements and athletic movements. The Organisers provided all competing groups with sensor data recorded on one leg above and below the knee. The training data set on which each team had to train their classifier contained sensor data collected from 15 subjects. Data collected from four other subjects were used by the organisers as test dataset to evaluate the solutions provided by each participating team.

For each of the subjects in both the training and the test dataset, participants were provided with data files containing the sensor reading data for various activities. One, particular activity is represented by data in a single data file where each line of the data file corresponds to the 19 sensor values measured at one time. Each sensor value is sampled at 1000 Hz which leads each column of the data file to represent a time series sensor value recording.

1.2 Method of Approach

The objective of our work in the Bremen Big Data Challenge is to find an optimal classifier to classify human activities based on the sensor readings provided to us. This optimal classifier, must be able to generalize its learning in the test data set with the smallest miss-classification rate we can achieve. With this goal set, our method of approach includes implementation of a full processing pipeline starting with feature extraction followed by using tree-based method for classification with subsequent steps where we also attempt to improve the performance of the classifier by careful filtration/selection of useful features and parameter tuning using Cross-Validation.

1.3 Result

After completing the data processing pipeline and training the classifier, the classifier was used to predict the activities present in the test dataset. After each run of training and predicting, the predictions were submitted to the organisers for performance evaluation of the classifier. The organizers evaluated the performance by using the recognition rate represented mathematically by,

$$Accuracy = \frac{numberofcorrectlyassignedlabels}{totalnumberoflabels} \tag{1}$$

Among the different models we trained, the best model has an prediction accuracy of 89.35%.

2 Introduction

Human activity monitoring has a broad range of applications which include old-age home care, prisoner monitoring, physical therapy/rehabilitation for athletes and public security [1]. In recent years the advancements in computational power and development of new statistical methods have enabled diverse applications of data-driven decision making which has increased the popularity and scope of research in human activity recognition using machine learning. The Bremen Big Data Challenge (BBDC) 2019, aimed to challenge students in training algorithms that predict various human activities with the least miss-classification rate . The challenge specifically aimed to recognize algorithms that could classify 22 different activities. The process of training such an algorithm entails implementation of a full processing pipeline from feature extraction to the training and tuning of classification algorithms.

In general, data used in recognizing human activity are typically sensor readings that are attached to the subject being monitored.[2] The data set provided by the BBDC 2019 organisers was sensor data recorded from sensors placed on one leg of the subject above and below the knee. The training data set contained sensor data of 15 subjects while the test data set contained sensor data from 4 other subjects. For each particular activity that was monitored, senors placed on the leg of each subject measure 19 different time series sensor values sampled at 1000 HZ.

In this project implemented a data processing pipeline to build a classification model which can be broken down into the following parts:

- Data Preprocessing step
- Algorithm Training
- Evaluation of algorithm performance
- Implementing approaches to increase algorithm performance

Following these steps we trained,evaluated and tuned our classification model using tree based algorithmic methods of classification. The performance of our model after subsequent training step was compared through the accuracy of their prediction on the test data-set.

In the initial prepossessing step we use the time series sensor readings provided to us and extract features we decided ,through literature review, that would be useful for the learning process of the classification algorithms. These features we extract remain the same for all activities preformed by the subjects in the training and testing dataset. Sections 3 and 4 of this report focus on describing the data provided by the organizers and the features we extract.

For the training step, we used the processed dataset and trained models using Decision Trees and Random Forest ensemble method with R-studio as our working platform. The training process involved subsequent steps of tuning the model for better performance. Section 5 of this report explains the choices we made in selecting the approach to train our classification model. Finally, section 6 of the report discusses the results of the various runs of predictions made by our classification model.

3 Dataset Description

As discussed earlier in the introduction section, The dataset provided by the BBDC 2019 organisers was sensor data recorded from sensors placed on one leg of the subject above and below the knee. Figure below shows the placement of the different sensors.

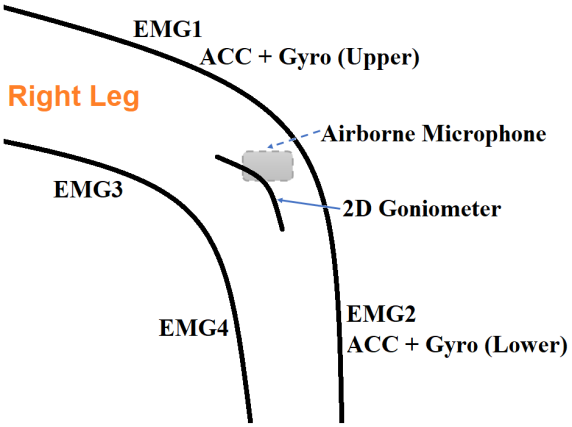


Figure 1: Sensor Placement

The sensors placed as such provide 19 different sensor values which are: 4 different Electromyography(EMG) recordings of the electrical activity produced by skeletal muscles, 1 Airborne microphone reading , 6 Acceleration readings of the upper and lower knee sensors (X,Y,Z components for each sensor), 2 Goniometer readings of the angle between the upper and lower knee (X and Y components), 6 Gyrometer readings for the rotational movements of the upper and lower knee sensors (X,Y,Z components for each sensor). Each of these 19 different sensor readings for a particular activity monitoring is sampled at a rate of 1000Hz i.e. for each activity monitored and for each of the 19 different sensor readings, we obtain a time series data where new readings are taken at an time interval of 1 milliseconds. The time series data for one activity obtained from these 19 different sensor readings was provided by the organisers as a data file where each line of the data file correspond to the 19 sensor values measured at one time. A total of 23 different activities were monitored these activities are:

- run
- walk
- stand
- sit-to-stand
- stand-to-sit
- stair-up
- stair-down
- jump-one-leg
- jump-two-leg
- curve-left-step
- curve-right-step
- curve-left-spin-Lfirst
- curve-left-spin-Rfirst
- curve-right-spin-Lfirst
- curve-right-spin-Rfirst
- lateral-shuffle-left
- lateral-shuffle-right
- v-cut-left-Lfirst
- v-cut-left-Rfirst
- v-cut-right-Lfirst
- v-cut-right-Rfirst
- lay
- sit

These activities were monitored for 19 subjects, among which data for 15 subjects were used in the training dataset and the remaining 4 in the testing dataset. The distribution of the number of times each subject preforms a particular activity is not uniform. The activity 'lay' in the dataset is present only for for subject 6 and had 23 distinct cases. We decided to drop this activity and work with the remaining 22. The training dataset was provided as a csv file where each line corresponds to the sensor value recording of a activity with columns that correspond to

- Subject: The ID of the subject
- Datafile: Path of the file containing the sensor data for this recording.
- Label: The movement that was recorded

	A	B	C
1	Subject	Datafile	Label
2	Subject02	Subject02/Subject02_Aufnahme000.csv	curve-left-step
3	Subject02	Subject02/Subject02_Aufnahme001.csv	curve-left-step
4	Subject02	Subject02/Subject02_Aufnahme002.csv	stand-to-sit
5	Subject02	Subject02/Subject02_Aufnahme003.csv	curve-right-spin-Rfirst
6	Subject02	Subject02/Subject02_Aufnahme004.csv	jump-one-leg
7	Subject02	Subject02/Subject02_Aufnahme005.csv	lateral-shuffle-right
8	Subject02	Subject02/Subject02_Aufnahme006.csv	curve-right-spin-Lfirst
9	Subject02	Subject02/Subject02_Aufnahme007.csv	v-cut-right-Lfirst
10	Subject02	Subject02/Subject02_Aufnahme008.csv	stair-down
11	Subject02	Subject02/Subject02_Aufnahme009.csv	v-cut-left-Rfirst
12	Subject02	Subject02/Subject02_Aufnahme010.csv	v-cut-right-Rfirst
13	Subject02	Subject02/Subject02_Aufnahme011.csv	jump-two-leg
14	Subject02	Subject02/Subject02_Aufnahme012.csv	sit
15	Subject02	Subject02/Subject02_Aufnahme013.csv	stair-up
16	Subject02	Subject02/Subject02_Aufnahme014.csv	curve-right-step
17	Subject02	Subject02/Subject02_Aufnahme015.csv	sit-to-stand
18	Subject02	Subject02/Subject02_Aufnahme016.csv	run
19	Subject02	Subject02/Subject02_Aufnahme017.csv	curve-right-spin-Lfirst
20	Subject02	Subject02/Subject02_Aufnahme018.csv	curve-right-step

Figure 2: The training file given by the BBDC, which shows us the subject, datafile and the label

The test data set was also provided in a similar format but with the label column left out, to be filled in with the predictions.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	32742	32901	32878	31973	33176	32351	33860	37308	34296	32916	37236	34429	28296	28632	29496	23821	30816	30013	23292
2	32828	32837	32941	32427	33499	32351	33860	37308	34296	32916	37236	34429	28296	28632	29496	23821	30816	30013	23292
3	32891	32645	32946	33031	33139	32351	33860	37308	34296	32916	37236	34429	28296	28632	29496	23821	30816	30013	23292
4	32967	32506	33010	32319	33208	32351	33860	37308	34296	32916	37236	34429	28296	28632	29496	23821	30816	30013	23292
5	32908	32375	32991	32193	32825	32351	33860	37308	34296	32916	37236	34429	28296	28632	29496	23821	30816	30013	23292
6	32920	32407	32998	33588	32375	32351	33860	37308	34296	32916	37236	34429	28296	28632	29496	23821	30816	30013	23292
7	32852	32872	33008	35781	31996	32351	33860	37308	34296	32916	37236	34429	28296	28632	29496	23821	30816	30013	23292
8	32748	33383	32981	35967	31426	32351	33860	37308	34296	32916	37236	34429	28296	28632	29496	23821	30816	30013	23292
9	32633	33581	33007	34189	31811	32351	33860	37308	34296	32916	37236	34429	28296	28632	29496	23821	30816	30013	23292
10	32680	33967	32867	31311	31599	32351	33860	37308	34296	32916	37236	34429	28296	28632	29496	23821	30816	30013	23292
11	32848	33115	32870	30555	32093	32201	33872	37386	34268	32765	37340	34392	28268	28508	29348	23040	30943	29604	21877
12	32877	31898	32731	31275	32258	32201	33872	37386	34268	32765	37340	34392	28268	28508	29348	23040	30943	29604	21877
13	32735	32593	32735	32027	33154	32201	33872	37386	34268	32765	37340	34392	28268	28508	29348	23040	30943	29604	21877
14	32714	32557	32763	32579	33565	32201	33872	37386	34268	32765	37340	34392	28268	28508	29348	23040	30943	29604	21877
15	32730	32559	32767	33203	33761	32201	33872	37386	34268	32765	37340	34392	28268	28508	29348	23040	30943	29604	21877
16	32675	32583	32751	33159	33510	32201	33872	37386	34268	32765	37340	34392	28268	28508	29348	23040	30943	29604	21877
17	32775	32679	32811	33302	32965	32201	33872	37386	34268	32765	37340	34392	28268	28508	29348	23040	30943	29604	21877
18	32853	32711	32797	33380	32562	32201	33872	37386	34268	32765	37340	34392	28268	28508	29348	23040	30943	29604	21877
19	32852	32646	32977	33112	33337	32201	33872	37386	34268	32765	37340	34392	28268	28508	29348	23040	30943	29604	21877
20	32805	32727	32918	33127	34294	32201	33872	37386	34268	32765	37340	34392	28268	28508	29348	23040	30943	29604	21877
21	32804	32752	32923	33198	34919	31972	33949	37372	34305	32532	37256	34400	28248	28584	29137	22760	30552	30186	21500

Figure 3: An example of all 19 sensor readings from the BBDC

4 Data Preprocessing: Feature Extraction and Exploration

Data preprocessing is the initial step of implementing any machine learning algorithm. As classification is a supervised learning task, both the training and testing data-set must be first represented as a labelled pair[3]:

$$(x_i, y_j) \tag{2}$$

where x_i are the different features that represent a labeled activity class y_j. To convert the training and testing dataset provided to us into labelled pairs, for each of the 19 sensor readings we extracted features (x_i) that we decided would be useful for the learning process of the classification algorithms. Next we represented each activity as a class($y_j \mid j \in (1, ..., 22)$). The process of data preprocessing was implemented in Python. The script used to implement this process can be referenced in appendix A. The subsections below explain the preprocessing plan and explores the processed dataset.

4.1 Class Labelling

We implemented the class labelling with a map corresponding to the mapping:

- 1 = run
- 2 = walk
- 3 = stand
- 4 = sit
- 5 = sit-to-stand
- 6 = stand-to-sit
- 7 = stair-up
- 8 = stair-down
- 9 = jump-one-leg
- 10 = jump-two-leg
- 11 = curve-left-step
- 12 = curve-right-step
- 13 = curve-left-spin-Lfirst
- 14 = curve-left-spin-Rfirst
- 15 = curve-right-spin-Lfirst
- 16 = curve-right-spin-Rfirst
- 17 = lateral-shuffle-left
- 18 = lateral-shuffle-right
- 19 = v-cut-left-Lfirst
- 20 = v-cut-left-Rfirst
- 21 = v-cut-right-Lfirst
- 22 = v-cut-right-Rfirst

Figure 4 shows the total count of each activity in the training dataset, it is clear from the dataset that the number of times each activity is represented in the dataset is not equally distributed. Extreme cases of class imbalance restrict the learning process of classification algorithms which we need not worry about. As evident from figure 4 although the representation of each class is not uniform we do not see any class that is underrepresented.

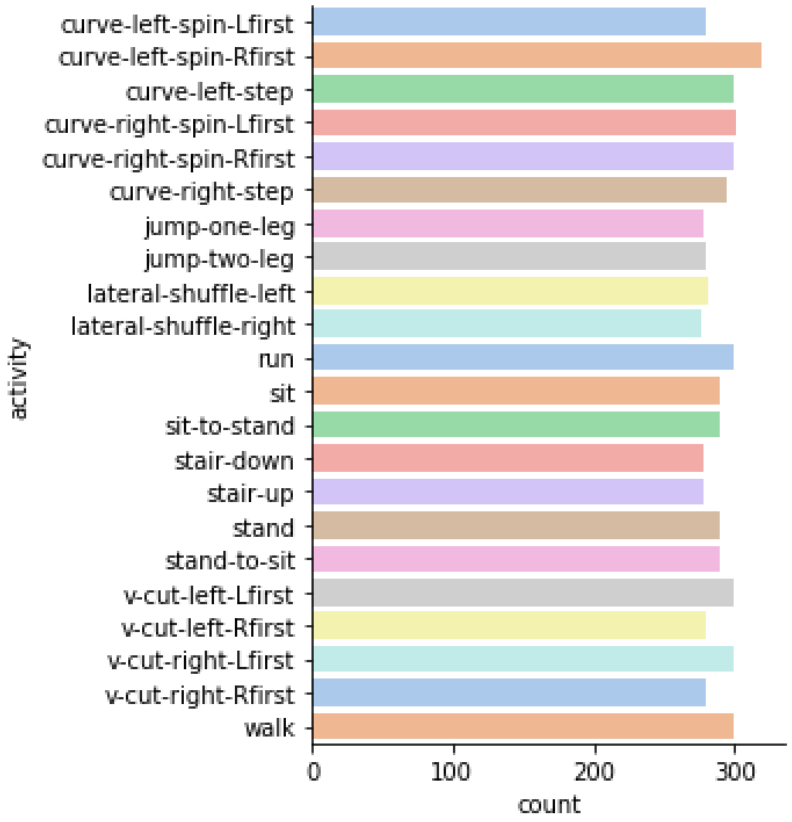


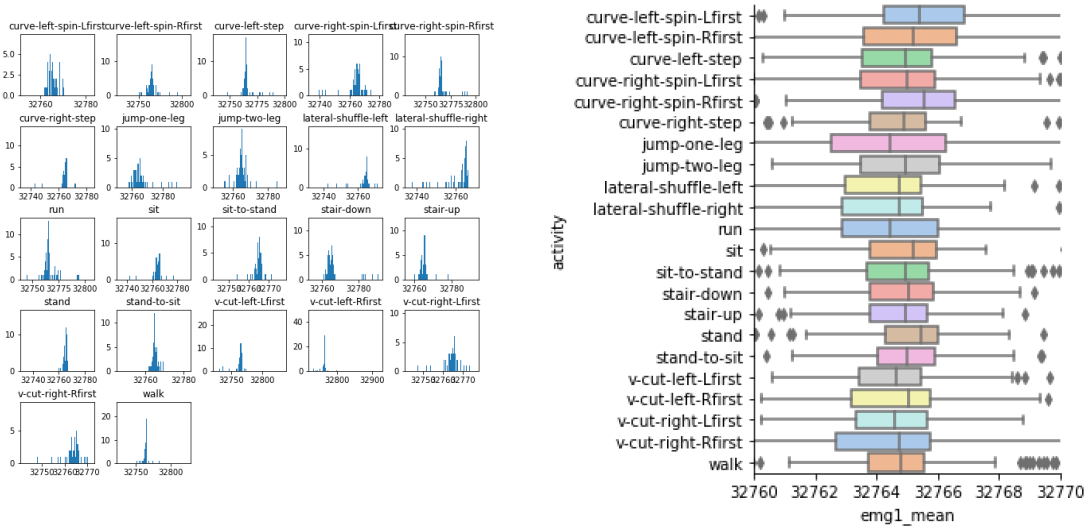
Figure 4: Count of different activities in the data set

4.2 Feature Extraction and Exploration

Feature extraction is seen as the first stage of any classification task as working with the raw data poses several problems in machine learning tasks[3]. For our specific task each activity preformed by a subject was provided to us as a data file where each line of the data file correspond 19 sensor values measured at one time with the 19 sensor values sampled at intervals of 1 milliseconds. The features we extract therefore had to be representative of the 19 different sensor readings which are in principle time series data. Looking at statistical characteristics of the sensor readings as features is a good start to begin training classification trees[4]. The number of features we extract remain the same for each activity preformed by all the subjects in the training and testing dataset. Graphically exploring the distributions of these features for different activities gives us an hint if the features will be useful for prediction or not. In this section we discuss the features we extracted and show example plots of the feature's distribution across different activities. Plotting graphs for each feature that we extract was not possible as the total number of features we extract is 19×8 so we plot a random feature example for each feature discussed in this section. The features we extracted are as follows:

- **Mean:**
  This feature would represent a particular sensor reading with its average value. It is expected that the classifier will be able to learn the difference between activities based on the mean value of the 19 sensor readings for each activity. It is also expected that the average sensor value recorded for a activity preformed by different subjects must lie within a particular range. Figure 5a traces the density plot for the mean value of the emg1 sensor. It is seen from the figure that distribution of the mean value recorded for the emg1 sensor for the different activities are not well separated for all the activities. But still some separation between different activities can be seen. These separation are seen more clearly in figure 5b where the distributions are plotted as box plots. even though clear separation in the distribution of the mean value of this particular sensor reading is not seen we still extract the mean value of each sensor as a feature which we will remove based on feature selection methods which we use to tune the algorithm later. Equation 3 shows how mean is calculated mathematically.

$$\overline{x} = \frac{1}{N}\sum_{i=1}^{N}(x_i) \tag{3}$$

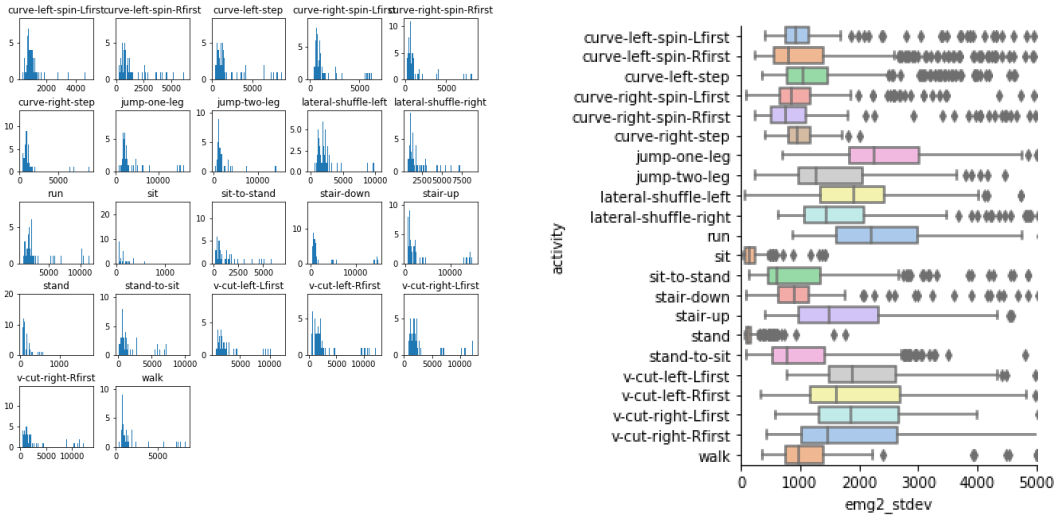


(a) density plot (b) box plot

Figure 5: distribution plots for the mean value of the emg1 sensor for different activities

- **Standard Deviation:**
  The standard deviation would show us the average spread of the sensor reading values of a particular sensor. We expect the classifier to be able to differentiate between activities based on the difference in distribution of the standard deviation values of the sensor readings.Figure 6 shows the distribution plot for the standard deviation value of the emg2 sensor. It is seen from figure 6b that distribution of this particular feature for the different activities are separable for most activities. One feature to notice in the plots is the number of out-lier points which can pose a problem for the algorithm's learning process. Equation 4 shows how standard deviation is calculated mathematically.

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{4}$$

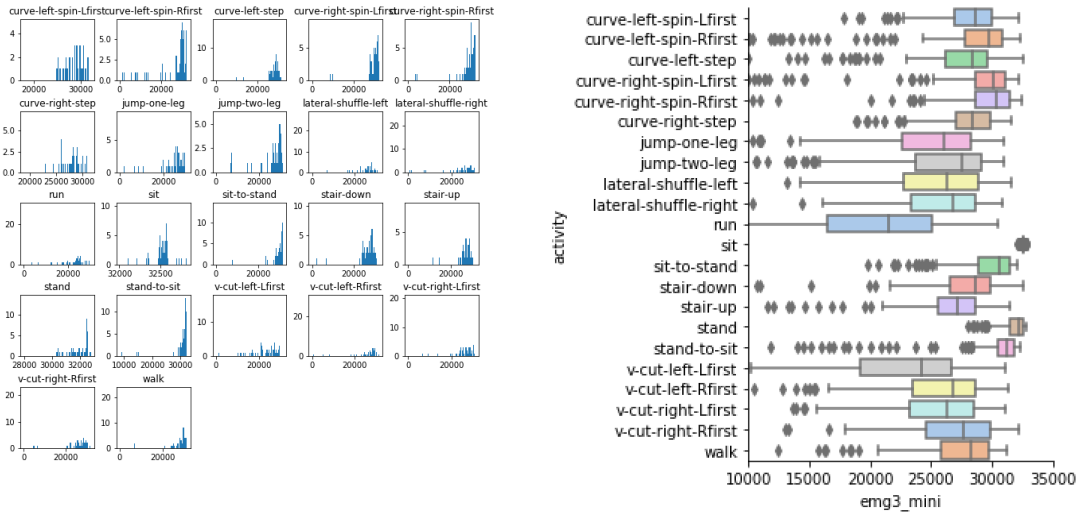


(a) density plot

(b) box plot

Figure 6: distribution of the standard deviation for the emg2 sensor readings for the different activities

- **Minimum:**
  This feature represents the minimum value of a sensor reading. Figure 7 shows the distribution plot for the minimum value of the emg3 sensor. It is seen from the figure that distribution of the minimum value for the emg3 sensor readings for the different activities are non-separable for most activities, but we still keep minimum value as a feature which we will remove based on feature selection methods which we use to tune the algorithm.

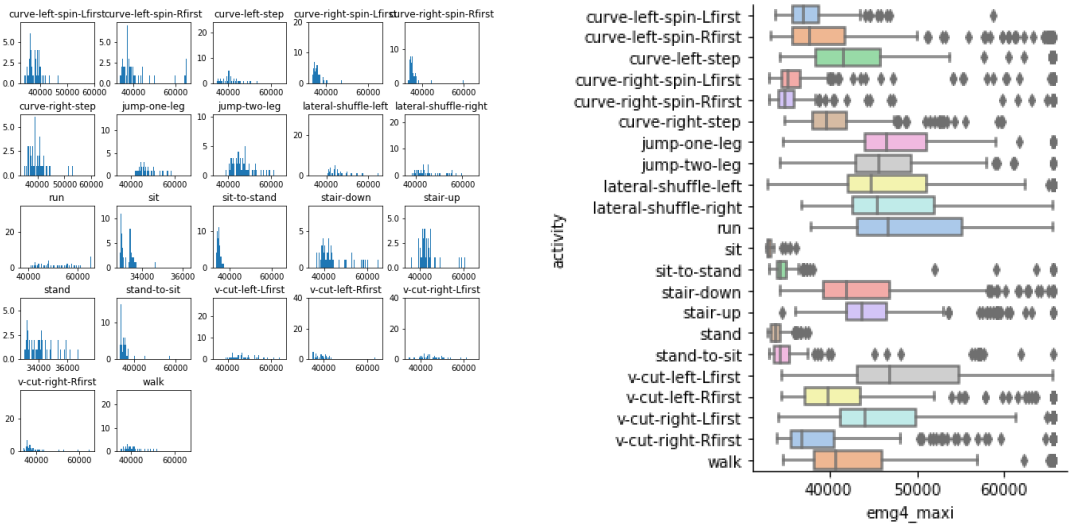


(a) density plot

(b) box plot

Figure 7: distribution of the minimum value for the emg3 sensor readings for the different activities

- **Maximum:**
  In contrast to the minimum value, this feature represents the maximum value of a sensor reading. Figure 8 shows the distribution plot for the maximum value of the emg4 sensor. It is seen from the figure that distribution of the maximum value for the emg4 sensor readings for the different activities are separable for some activities.

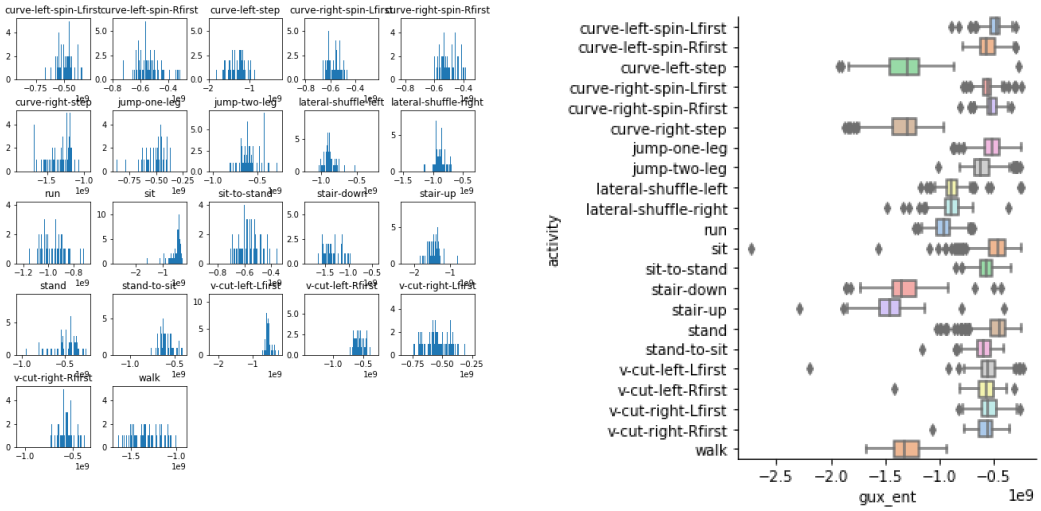


(a) density plot

(b) box plot

Figure 8: distribution of the maximum value for the emg4 sensor readings for the different activities

- **Entropy:**
  This feature helps us quantify the amount of regularity and unpredictability of the sensor readings. Figure 9 shows the distribution plot for the entropy of the Upper Gyrometer sensor reading in the X-axis. It is seen from the figure that distribution of the entropy is separable for some activities.



(a) density plot

(b) box plot

Figure 9: distribution of the entropy of the Upper Gyrometer sensor reading in the X-axis for the different activities

- **Auto regressive model Value:**
  An auto regressive model value is obtained when a value from a time series is regressed on previous values from that same time series. This feature in its core would represent a predicted sensor value for each sensor. Figure 10 shows the distribution plot for the auto regressive model value of the Gyrometer sensor reading in the y-axis for the different activities. It is seen from the figure that distribution of these auto regressive model value are non-separable for most activities, but we still keep this as a feature which we will remove based on feature selection methods which we use to tune the algorithm.

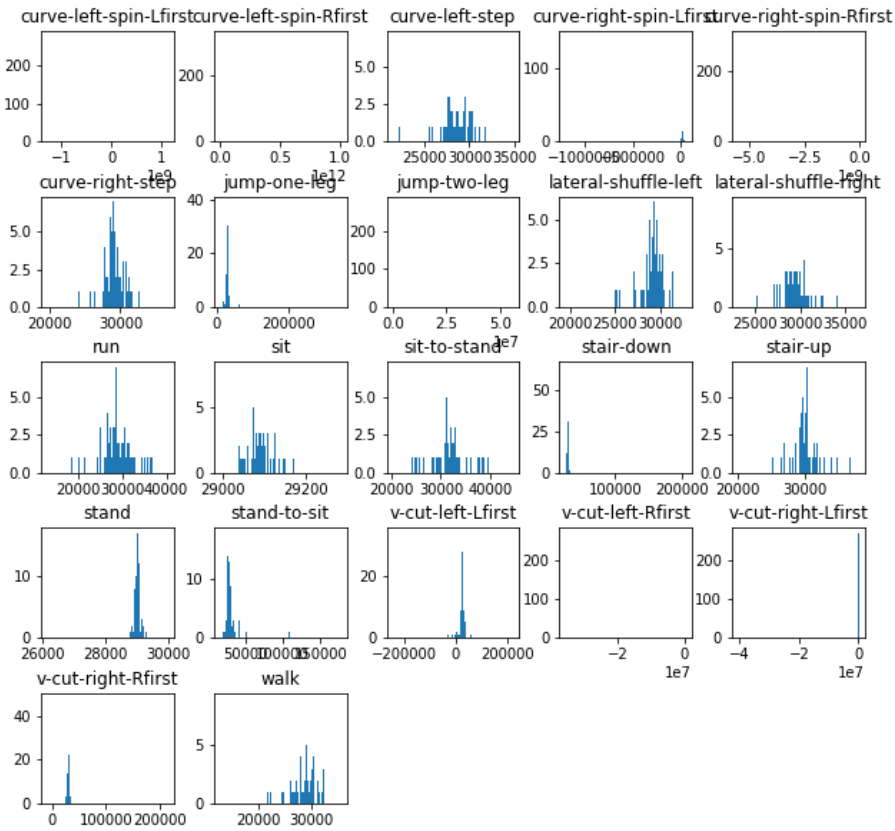
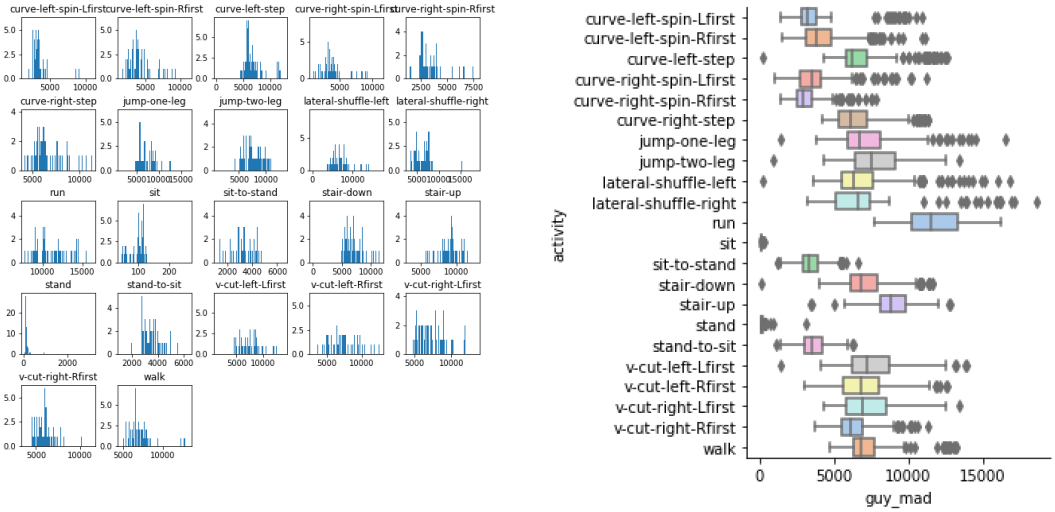


Figure 10: distribution of the auto regressive model value of the Gyrometer sensor reading in the y-axis for the different activities

- **Median Absolute Deviation:**
  This feature would represent a measure of the variability of the sensor readings. Figure 11 shows the distribution plot for the Median absolute deviation of the lower Gyrometer sensor reading in the Y-axis. It is seen from the figure these distributions are separable for most activities. Equation 5 shows how median absolution deviation is calculated mathematically.

$$MAD = \frac{\sum || \, x_i - \overline{x} \, ||}{n} \tag{5}$$

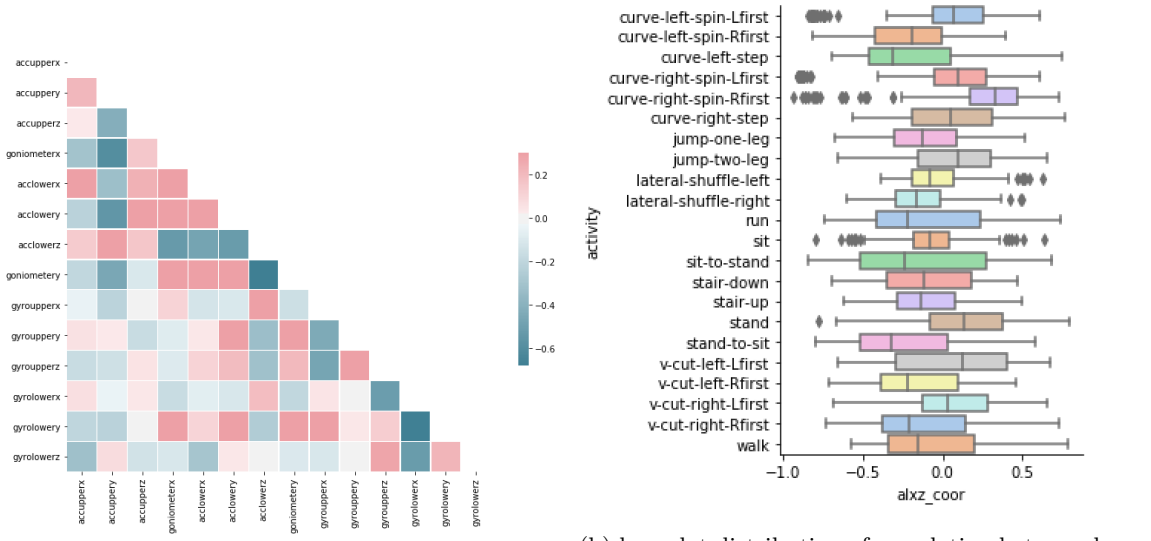


(a) density plot

(b) box plot

Figure 11: distribution of the Median absolute deviation of the lower Gyrometer sensor reading in the Y-axis for the different activities

- **Correlation:**
  This feature represents the correlation between different sensor values. This feature is especially important for sensors that give values in different coordinate axis. It will give us an idea if the values in the different coordinate axis are correlated to each other, it is expected that for activities in which muscle movement happens in different coordinate directions the sensor values will be correlated[5]. For any particular activity, this feature also will also help the classifiers learn if any of the sensor readings are correlated. Figure 12 shows the correlation matrix plot for the 19 sensor readings for the activity "curve-left-step". It is seen from the figure that some of the sensor readings, especially the sensor readings split in the XYZ-coordinate axis are correlated to each other.



(a) the correlation matrix plot for the 19 sensor readings for the activity "curve-left-step"

(b) box plot distribution of correlation between lower acceleration senor reading in the X and Z directions for the various activities.

Figure 12: plots exploring correlation feature

- **Inter-quartile Range:**
  This feature represents a statistical measure which is equal to the difference between 75^{th} and 25^{th} percentiles, or in plain words lies between upper and lower quartiles. It can be represented as follows:

$$\mathcal{Q} : \mathcal{Q}_3 - \mathcal{Q}_1 \tag{6}$$

5 Data Analysis

5.1 Tree based classification models

We built the basic model of classification using Decision tree as our classifier. The aim of this basic model was to see how well the features we extracted help classification trees to classify the activities. Running this initial model we got an training accuracy of 71%. Figure below shows the summary statistics of the predictions done on the training dataset.

```
Overall Statistics

               Accuracy : 0.7108
                 95% CI : (0.6995, 0.7219)
    No Information Rate : 0.05
    P-Value [Acc > NIR] : < 2.2e-16
```

Figure 13: training prediction evaluation

To learn how this model generalizes while using the test data, we submitted the predictions made from the test dataset to the BBDC organisors. The test prediction accuracy we got was 67%. This was an indicator for us that, our model was over fitting in the training data and using a single tree would not be the best option for a multi-class classification with 22 classes[6]. To remedy this we decided on using ensemble of trees and averaging their predictions using the random forest approach. Using an ensemble of decision trees acts as an regularizer which restricts variance of classification trees ultimately helping us prevent overfitting on the training data.

5.2 Random Forest

The random forest classifier fits an ensemble of decision trees to random samples of the data provided. Further when fitting each tree the algorithm only considers a randomly generated subset of features for each node and split. In addition to considering randomized subset of features for each split, decision trees have an inbuilt algorithmic of comparing the impact each feature in the randomized subset has at the split and choosing the feature with the most impact[6]. This allowed us to add more features in the dataset later namely inter-quartile range and correlation between variables to achieve better prediction accuracy.

The number of trees to grow in the forest was a decision we could also possibly use cross-validation to decide. But, with limited computational power to run a full K fold cross validation grid search for the optimal number of trees grown, we decided run a random forest model with 200 trees and decide a suitable number of trees to use by comparing the Out Of Bag (OOB) error rate calculated by the model for each number of tree grown. While doing so we noticed that the number of trees grown does not significantly decrease the error rate past a certain point. we did not achieve any significant change in the OOB error rate when

using more than 150 trees. Figure below shows the OOB error plotted against the number of trees.

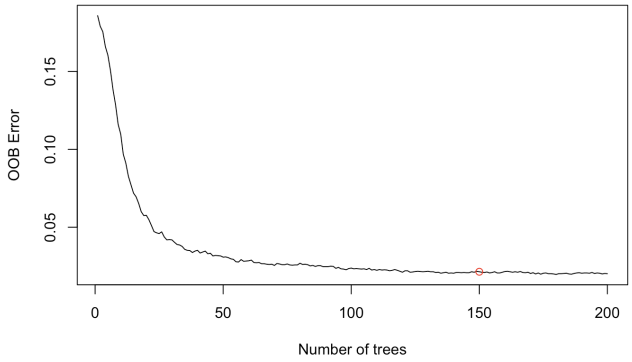


Figure 14: plot of the OOB error rates against number of trees grown

The next parameter we could tune in the tree based model was the Number of features randomly sampled as candidates at each split. For this we ran a k = 5 fold cross validation on a random forest model with 150 trees with a a grid search for the number of features between 10-15. Most literature on the Random forest suggest that the ideal parameter value to use is $\sqrt{p}$ where p is the number of features. For us this value would be approximately 11, which is why we decided to do a grid search in the range 10-15. Figure below shows the plot of the cross validation errors versus the number of feature candidates for split.

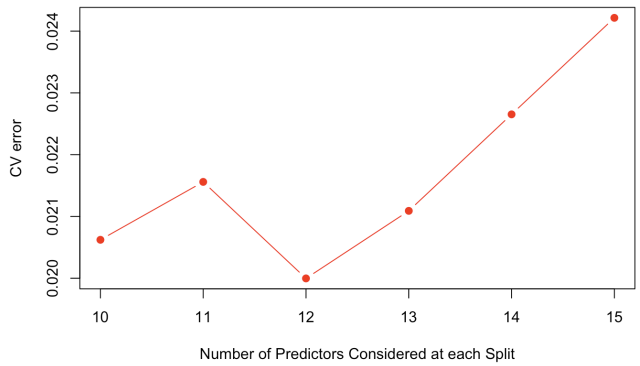


Figure 15: plot of the Cross validation error rates against number of feature candidates for split.

6 Results and Conclusion

date	Score
2019-03-30 20:56:59	0.8878020713463751
2019-03-30 20:30:07	0.8935558112773303
2019-03-30 20:17:18	0.8624856156501726
2019-03-30 19:56:58	0.8624856156501726
2019-03-30 16:53:45	0.8636363636363636
2019-03-24 19:01:49	0.8693901035673187
2019-03-24 18:02:40	0.8659378596087457
2019-03-24 16:32:01	0.6737629459148446
2019-03-23 21:12:15	0.17261219792865362

Figure 16: Submissions made on the BBDC website

Our first submission using a single classification tree yielded accuracy of 17.26%. Which was surprising for us as a low accuracy was not what we expected. While reviewing the code we used to generate numerical labels for the the activities we discovered that the code had labelled the activities curve-left-spin-Lfirst, curve-left-spin-Rfirst, curve-right-spin-Lfirst and curve-right-spin-Rfirst with the same numerical value. The second submission we made was again with same single classification tree with corrected class labelling. This model was able to achieve an accuracy of 67.37% which was a decent result considering the use of a single classification tree without any parameter tuning and pruning.

The third submission we made was using the predictions of a Random Forest model with 50 trees and 10 feature candidates for each split. This model achieved a test accuracy of 86.5%. The fourth submission we made was with the predictions form a Random Forest model with 150 trees and 12 feature candidates for each split i.e. using the parameter values we choose by comparing the OOB error rate and crossvalidation respectively. The model achieved an accuracy of 86.9%.

Looking at the small change in the accuracy, after using the optimal parameters we selected, we decided to check if the accuracy changed when we train the classifier with a limited number of feature after feature selection. Following this idea we decided to use a subset of features by selecting features on the basis of feature importance given by the random forest model. We then fit two random forest model using only 120 features first and 100 features next instead of the original 134 features, keeping the number of trees and the number of split variables constant. The prediction of these model were our fifth and sixth submissions. We noticed that reducing the number of feature decreased our prediction accuracy. We also realized that since, decision trees compares the impact each possible feature has at a split and chooses the feature with the most impact including more features would be more beneficial than excluding features. Therefore we decided to extract two more features for each sensor, namely inter-quartile range and correlation between variables, resulting in 36 new features to be added in the training and test dataset. We then built our final model using the optimal parameters

of 150 trees and 12 feature candidates for each split. We trained and tested this particular model using the new dataset with additional features. This model was able to achieve the highest test accuracy of 89.35%.

In conclusion we have learnt that the random forest classification technique can be a good start for building models that classify human activities. Furthermore, we believe that the models we built were unable to reach high prediction rates due to the simplified nature of our extracted features. The features we extracted were simple statistical characteristics of the sensor readings and were not based on derived properties of the wavelets/signal generated by the sensors. This could have potentially introduced a high amount of bias in the models we built restricting higher performance.

7 References

References

[1] T. P. 'P Dohnalek', 'P Gajdos' and V. Snasel', *An Overview of Classification Techniques for Human Activity Recognition.* 2013-11-1.

[2] V. B. "Terry Taewoong Um" and D. Kulic, *Exercise Motion Classification from Large-scale Werable Sensor Data Using Convolutional Neural Networks.* 2017-07-22.

[3] H. Jaeger, *Machine Learning: Lecture Notes.* 2019-02-04.

[4] A. Wilhelm, *Assignment 7 Data Analytics-Fall 2018 : Human Activity Recognition using Smartphones Dataset.* 2018-04-11.

[5] V. K. "Ankita Jain", *Human Activity Classification in Smartphones Using Accelerometer and Gyroscope Sensors.* 2018-02-01.

[6] J. F. "Trevor hastie", "Robert Tibshirani", *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* 2001.

A Code

A.1 Feature Extraction implemented in Python

```
import numpy as np
import pandas as pd
from scipy.special import entr
from pandas import Series
from statsmodels.tsa.ar_model import AR
import xlsxwriter


def mean(value):
    return np.mean(value)

def stdev(value):
    return np.std(value)

def mini(value):
    return np.min(value)

def maxi(value):
    return np.max(value)

def ent(value):
    return entr(value).sum()

def arcoff(value):
    ser = pd.Series(value)
    train, test = ser[1:int(len(ser)*0.6)], ser[int(len(ser)*0.6):]
    model = AR(train)
    model_fit = model.fit()
    predictions = model_fit.predict(start=len(train), end=len(train)+len(test)-1, dynamic=False)
    return np.average(predictions)

def m_a_d(value):

    series = pd.Series(value)
    return series.mad()

data_train = pd.read_csv("challenge.csv")
train_matrix = data_train.as_matrix()
file_path = np.array(train_matrix[0:400,1])
x = file_path.shape[0]
for i in range(x):
```

```
reading_val = pd.read_csv(file_path[i],header=None)
value_matrix = reading_val.as_matrix()
r,c = value_matrix.shape

emg1 = np.array(value_matrix[0:r,0])
emg1_mean.append(mean(emg1))
emg1_stdev.append(stdev(emg1))
emg1_mini.append(mini(emg1))
emg1_maxi.append(maxi(emg1))
emg1_ent.append(ent(emg1))
emg1_arcoff.append(arcoff(emg1))
emg1_mad.append(m_a_d(emg1))


emg2 = np.array(value_matrix[0:r,1])
emg2_mean.append(mean(emg2))
emg2_stdev.append(stdev(emg2))
emg2_mini.append(mini(emg2))
emg2_maxi.append(maxi(emg2))
emg2_ent.append(ent(emg2))
emg2_arcoff.append(arcoff(emg2))
emg2_mad.append(m_a_d(emg2))

emg3 = np.array(value_matrix[0:r,2])
emg3_mean.append(mean(emg3))
emg3_stdev.append(stdev(emg3))
emg3_mini.append(mini(emg3))
emg3_maxi.append(maxi(emg3))
emg3_ent.append(ent(emg3))
emg3_arcoff.append(arcoff(emg3))
emg3_mad.append(m_a_d(emg3))

emg4 = np.array(value_matrix[0:r,3])
emg4_mean.append(mean(emg4))
emg4_stdev.append(stdev(emg4))
emg4_mini.append(mini(emg4))
emg4_maxi.append(maxi(emg4))
emg4_ent.append(ent(emg4))
emg4_arcoff.append(arcoff(emg4))
emg4_mad.append(m_a_d(emg4))

airborne = np.array(value_matrix[0:r,4])
ab_mean.append(mean(airborne))
ab_stdev.append(stdev(airborne))
ab_mini.append(mini(airborne))
ab_maxi.append(maxi(airborne))
```

```
ab_ent.append(ent(airborne))
ab_arcoff.append(arcoff(airborne))
ab_mad.append(m_a_d(airborne))


accupperx = np.array(value_matrix[0:r,5])
aux_mean.append(mean(accupperx))
aux_stdev.append(stdev(accupperx))
aux_mini.append(mini(accupperx))
aux_maxi.append(maxi(accupperx))
aux_ent.append(ent(accupperx))
aux_arcoff.append(arcoff(accupperx))
aux_mad.append(m_a_d(accupperx))



accuppery = np.array(value_matrix[0:r,6])
auy_mean.append(mean(accuppery))
auy_stdev.append(stdev(accuppery))
auy_mini.append(mini(accuppery))
auy_maxi.append(maxi(accuppery))
auy_ent.append(ent(accuppery))
auy_arcoff.append(arcoff(accuppery))
auy_mad.append(m_a_d(accuppery))

accupperz = np.array(value_matrix[0:r,7])
auz_mean.append(mean(accupperz))
auz_stdev.append(stdev(accupperz))
auz_mini.append(mini(accupperz))
auz_maxi.append(maxi(accupperz))
auz_ent.append(ent(accupperz))
auz_arcoff.append(arcoff(accupperz))
auz_mad.append(m_a_d(accupperz))

goniometerx = np.array(value_matrix[0:r,8])
gox_mean.append(mean(goniometerx))
gox_stdev.append(stdev(goniometerx))
gox_mini.append( mini(goniometerx))
gox_maxi.append(maxi(goniometerx))
gox_ent.append(ent(goniometerx))
gox_arcoff.append(arcoff(goniometerx))
gox_mad.append(m_a_d(goniometerx))


acclowerx= np.array(value_matrix[0:r,9])
alx_mean.append(mean(acclowerx))
```

```
alx_stdev.append(stdev(acclowerx))
alx_mini.append(mini(acclowerx))
alx_maxi.append(maxi(acclowerx))
alx_ent.append(ent(acclowerx))
alx_arcoff.append(arcoff(acclowerx))
alx_mad.append(m_a_d(acclowerx))

acclowery = np.array(value_matrix[0:r,10])
aly_mean.append(mean(acclowery))
aly_stdev.append(stdev(acclowery))
aly_mini.append(mini(acclowery))
aly_maxi.append(maxi(acclowery))
aly_ent.append(ent(acclowery))
aly_arcoff.append(arcoff(acclowery))
aly_mad.append(m_a_d(acclowery))

acclowerz= np.array(value_matrix[0:r,11])
alz_mean.append(mean(acclowerz))
alz_stdev.append(stdev(acclowerz))
alz_mini.append(mini(acclowerz))
alz_maxi.append(maxi(acclowerz))
alz_ent.append(ent(acclowerz))
alz_arcoff.append(arcoff(acclowerz))
alz_mad.append(m_a_d(acclowerz))

goniometery= np.array(value_matrix[0:r,12])
goy_mean.append(mean(goniometery))
goy_stdev.append(stdev(goniometery))
goy_mini.append( mini(goniometery))
goy_maxi.append(maxi(goniometery))
goy_ent.append(ent(goniometery))
goy_arcoff.append(arcoff(goniometery))
goy_mad.append(m_a_d(goniometery))

gyroupperx= np.array(value_matrix[0:r,13])
gux_mean.append(mean(gyroupperx))
gux_stdev.append(stdev(gyroupperx))
gux_mini.append(mini(gyroupperx))
gux_maxi.append(maxi(gyroupperx))
gux_ent.append(ent(gyroupperx))
gux_arcoff.append(arcoff(gyroupperx))
gux_mad.append(m_a_d(gyroupperx))

gyrouppery = np.array(value_matrix[0:r,14])
guy_mean.append(mean(gyrouppery))
guy_stdev.append(stdev(gyrouppery))
```

```
    guy_mini.append(mini(gyrouppery))
    guy_maxi.append(maxi(gyrouppery))
    guy_ent.append(ent(gyrouppery))
    guy_arcoff.append(arcoff(gyrouppery))
    guy_mad.append(m_a_d(gyrouppery))

    gyroupperz = np.array(value_matrix[0:r,15])
    guz_mean.append(mean(gyroupperz))
    guz_stdev.append(stdev(gyroupperz))
    guz_mini.append(mini(gyroupperz))
    guz_maxi.append(maxi(gyroupperz))
    guz_ent.append(ent(gyroupperz))
    guz_arcoff.append(arcoff(gyroupperz))
    guz_mad.append(m_a_d(gyroupperz))

    gyrolowerx = np.array(value_matrix[0:r,16])
    glx_mean.append(mean(gyrolowerx))
    glx_stdev.append(stdev(gyrolowerx))
    glx_mini.append(mini(gyrolowerx))
    glx_maxi.append(maxi(gyrolowerx))
    glx_ent.append(ent(gyrolowerx))
    glx_arcoff.append(arcoff(gyrolowerx))
    glx_mad.append(m_a_d(gyrolowerx))

    gyrolowery = np.array(value_matrix[0:r,17])
    gly_mean.append(mean(gyrolowery))
    gly_stdev.append(stdev(gyrolowery))
    gly_mini.append(mini(gyrolowery))
    gly_maxi.append(maxi(gyrolowery))
    gly_ent.append(ent(gyrolowery))
    gly_arcoff.append(arcoff(gyrolowery))
    gly_mad.append(m_a_d(gyrolowery))

    gyrolowerz = np.array(value_matrix[0:r,18])
    glz_mean.append(mean(gyrolowerz))
    glz_stdev.append(stdev(gyrolowerz))
    glz_mini.append(mini(gyrolowerz))
    glz_maxi.append(maxi(gyrolowerz))
    glz_ent.append(ent(gyrolowerz))
    glz_arcoff.append(arcoff(gyrolowerz))
    glz_mad.append(m_a_d(gyrolowerz))



workbook = xlsxwriter.Workbook('train.xlsx')
worksheet = workbook.add_worksheet()
```

A.2 Decision trees and random forest models implemented in R

```
#Decision Trees

library(rpart)
library(caret)


#Single Decision Tree model
df= training_data
df['activity_label'==23] <- NA
df = na.omit(df)

summary(df$activity_label)
test_data= test
tree_classifier = rpart(df$activity_label ~ ., data = df,method="class")
summary(tree_classifier)
plot(tree_classifier)
text(tree_classifier ,pretty=0)
train_pred = predict(tree_classifier,df,type="class")
test_pred = predict(tree_classifier,test_data,type="class")


#missclassification training

xtab=table(train_pred,df$activity_label)
confusionMatrix(xtab,mode="everything")

#pruning
pfit= prune(tree_classifier, cp=tree_classifier$cptable[which.min(tree_classifier$cptable[,"xerror"]
plot(pfit)
text(pfit ,pretty=0)
test_pred_prune = predict(pfit,test_data,type="class")
test_pred_prune
test_data$prediction_prune = test_pred_prune

#exporting dataframe to csv
write.table(test_data, "my_data_pred.csv", sep=",")

#Random Forest

library(randomForest)
library(MASS)
```

```
library(caret)

dfn=training_data
df_test = test

# Make dependent variable as a factor (categorical)
dfn$activity_label= as.factor(dfn$activity_label)
set.seed(100)
randomforest_c = randomForest(dfn$activity_label~.,data=dfn,mtry=11,ntree=150,importance=TRUE)

##OOB error vs number of trees

oob = randomforest_c$err.rate
randomforest_c$err.rate
plot(seq(1,200),oob[1:200,1],type = "l",xlab='Number of trees',ylab='OOB Error')
points(150,0.02140291,type="o",col='red')

####find best mtry with CV

rf.cv = randomForest.crossvalidation(randomforest_c,cv.fold=5)

cv.err<-double(5)

#mtry is no of Variables randomly chosen at each split
for(mtry in 10:15)
{
  rf.cv = randomForest.crossvalidation(randomforest_c,cv.fold=5)
  oob.err[mtry] = rf$err.rate[200,1]
  cat(mtry," ")
}

cv.err[10:15]
matplot(10:15 , cv.err[10:15], pch=19 , col="red",type="b",ylab="CV error",xlab="Number of Predictor

#predict
test_prediction_rf_n = predict(randomforest_c,df_test,type="class")


df_test$Label = test_prediction_rf_n

write.table(df_test, "my_data_rf.csv", sep=",")
```