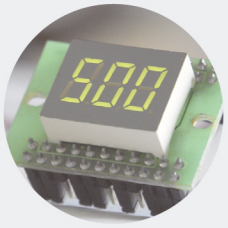
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






MOBILE ORIENTATION SYSTEM BASED ON INERTIAL SENSORS

GIRISH KANIKA
novembro de 2018

POLITÉCNICO
DO PORTO

MOBILE ORIENTATION SYSTEM BASED ON INERTIAL SENSORS

GIRISH KANIKA

1161905

Master's in Electrical and Computer Engineering
Area of Specialization in Automation and Systems
Department of Electrical and Computer Engineering
Instituto Superior de Engenharia do Porto
2018

This report satisfies, in part, the requirements contained in the Thesis Discipline Sheet of the 2nd year of the MSc in Electrical and Computer Engineering

Candidate: Girish Kanika, Nº 1161905, 1161905@isep.ipp.pt

Supervisor: Lino Figueiredo, lbf@isep.ipp.pt

Company: Ubirider

Supervision: Jorge Pinto, jorgepinto@ubirirder.com

MSc in Electrical and Computer Engineering

Area of Specialization in Automation and Systems

Department of Electrical Engineering

Instituto Superior de Engenharia do Porto

2018

Acknowledgements

I would like to express my gratitude to Dr. Lino Figueiredo for his full support, expert guidance, understanding and his incredible patience throughout project development.

I would also like to thank my supervisors Paulo Ferreira dos Santos (CEO), Jorge Pinto (CTO), Diogo Santos (Android Developer) and the rest of the team for all the help, support, guidance and integration in the company, as well as knowledge shared during the internship.

Thanks also to my friends for the good times, for all the support, and specially friendship. Finally, I would like to thank my parents, my brother, sister and all the family for their unconditional love and support during all my studies.

Abstract

Mobile devices have become a part of digital livelihood. Every user carries phones with them throughout the day as they drive, walk, run and work. Understanding users doing in the physical world allows the smarter way of interacting. The Activity Recognition is built on top of the sensors available in a device. These sensors include GPS sensors, temperature sensors, direction sensors (i.e., magnetic compasses), and acceleration sensors (i.e., accelerometers). This paper describes and evaluates a system that uses phone-based accelerometers to perform activity recognition analysis, a task which involves recognition of on-board vehicles that runs in underground environment (without GPS) and identifying the metro starting, moving and stopping accelerations. To implement the system the data collected labelled accelerometer data from arrival to destination stops and then aggregated the time series data in different positions into samples. An algorithm is developed using machine learning predictive analysis.

Keywords: Activity Recognition, Sensors, Neural Networks GPS, Android, Accelerometer, Machine Learning. Classification.

Resumo

Os dispositivos móveis tornaram-se parte da subsistência digital. Cada utilizador transporta consigo os telefones durante o dia enquanto dirige, anda, corre e trabalha. Entender o que os utilizadores fazem no mundo físico é a forma mais inteligente de interação. A Activity Recognition é determinada com base nos sensores disponíveis num dispositivo móvel. Estes sensores incluem sensores GPS, sensores de temperatura, sensores de direção (bússolas magnéticas) e sensores de aceleração (acelerómetros). Este trabalho descreve e avalia um sistema que usa o acelerômetro, disponível no telefone, para realizar a análise de reconhecimento de atividade, uma tarefa que envolve o reconhecimento da catividade de veículos que correm em ambiente subterrâneo (sem GPS) e identificando as acelerações de partida, movimentação e paragem do metro. Para implementar o sistema, foram recolhidos os dados do acelerômetro ao longo de um percurso do metro, em seguida, foram agregados os dados da série temporal em diferentes posições nas amostras. Por fim foi implementado um algoritmo usando uma análise preditiva baseada no conceito de Machine Learning.

Palavaras-chave: Activity Recognition, Sensors, Neural Networks GPS, Android, Accelerometer, Machine Learning.

Index

Contents

Table of Figures

List of Tables

Abbreviations

AUC	-	Area Under Curve
DT	-	Decision Trees
DTW	-	Dynamic Time Wrapping
FPR	-	False Positive Rate
GPS	-	Global Positioning System
MLP	-	Multi-Layer Perceptron
NB	-	Naïve Bayes
PCA	-	Principal Component Analysis
RF	-	Random Forest
ROC	-	Receiving Operating Characteristics
SVM	-	Support Vector Machine
TPR	-	True Positive Rate
WEKA	-	Waikato Environment for Knowledge Analysis

1 Introduction

Mobile devices have a large impact in human daily life activities. The smartphones growth in the market is high which are incorporated with diverse and powerful sensors. These sensors include GPS sensors, audio sensors (*i.e.*, microphones), light sensors, temperature sensors, direction sensors (*i.e.*, compasses) and acceleration sensors (*i.e.*, accelerometers). Understanding what users are doing in the physical world allows the app to be smarter and interact with user. For example, an app can start tracking a user's heartbeat when starts running, another app can switch to car mode when it detects driving. Because of the small size of these "smart" devices, their substantial computing power, send and receive data these are open to data mining research applications like financial banking and customer relationship management.

In the smartphone, accelerometer has been mainly used for the activity recognition. Some people have used gyroscope too. For the best analysis these two sensors are not individually analysed, instead an accelerometer is considered as the main sensor because of the performance based on orientation, position and previous experiments carried out and a gyroscope as an additional sensor in activity recognition [1].

1.1 Context

This project arose from the Ubirider a startup company to develop a universal application for smartphone to plan a trip and make the payment in advance in all modes of public transport and car parks. The project theme is based on using the algorithm of recognition and categorization of everyday activities (walk, run etc) and develop a recognition displacement of on-board vehicles without using GPS which is unavailable in non-existent places. This led to the development of algorithm in the application on this thesis, mobile orientation system based on inertial sensors.

Ubirider is a start-up company located in Porto, Portugal developing a universal application called Ubirider used for urban mobility to plan a trip and make advance payment in all modes of public transport and car parking. Ubirider features provide information on public transportation operators, cities and local places on the go.

1.2 Goals

In this project we explore the use of sensor like accelerometer, to identify the user on board activity in vehicles without GPS. Finally, an algorithm is developed to integrate in Android and IOS applications. A triaxial accelerometer is a sensor that returns an estimate of

acceleration along the x, y, z axes from which velocity and displacement can also be estimated.

Activity recognition is formulated as a supervised classification problem, whose training data is obtained via collection of multiple samples of on-board vehicles.

The main goal of this internship is to develop an algorithm for Android/IOS, that runs in underground environment, identifying the passenger activity during the on board of a metropolitan. The algorithm should support the following activities like

Starting and stopping times of the metropolitan

1.3 Approach

In this project the approach starts with collecting the data. The data is processed onto machine learning with pre-processing steps and applying various neural networks. There are different neural networks like SVM, Random Forest etc., on which the data analysis is done. The neural network with high accuracy and less processing time is considered for implementing in the algorithm.

1.4 Timeline

The Schedule of the project was planned by the author according to table a, available in Appendix A. In this table it can be observed how the project was planned. Since it was developed in company environment, some changes have occurred from the project planned, however all the stages have been accomplished.

1.5 Structure

This section presents the structure of the report, consisting of the following seven chapters:

The first chapter is an introductory chapter which talks about the goals, timeline, schedule activity recognition and sensors used in the project.

In the second chapter discusses about the state of art which consists of different references and the projects accomplished in the activity recognition.

Third chapter consists of the machine learning process, neural networks used for developing the machine learning algorithms.

The fourth chapter talks about the technology used. The project is completely developed in software discusses about MATLAB, WEKA and Android Studio.

The fifth chapter discusses about the data collection process and the preparation of the experiment.

The sixth chapter presents implementation of the project and the results obtained in the project.

In the seventh chapter presents conclusion of the work and future work development to improve the accuracy of the system.

2 State of the Art

This chapter presents a literature survey to systems that may have similar characteristics to the project in question. A study of some physical activities that may be useful during the development of the project has also been carried out.

Based on this research papers, on the physical activity level technologies founded and the existing project, a description of some key issues for an easy understanding of the choices made during the project development is also shown. In many works have addressed the problem of activity recognition from accelerometer data [2] produced by wearable devices. Moreover, the methodology adopted studies the accelerometer data from large set of samples, extract the features from raw data and utilize learning-based classification techniques to recognize the accelerometer pattern.

2.1 Towards Physical Activity Recognition Using Smartphone Sensors

The project details the analysis about the individual and combined roles of an accelerometer, a gyroscope and a magnetometer in physical activity recognition. This project better answers the question that in which situation an accelerometer performs better than a gyroscope and vice versa and when to combine the two sensors for better performance?

2.1.1 Experiment Design

The process gets initiated with the data collection from the smartphones. The android application collects data from an accelerometer, a magnetometer and a gyroscope at 50 samples per second. In the process of collecting data disturbances occur during the start and stop of each activity. These noisy parts were removed before filtering the data. Each sensor reports values along its three dimensions. Most of the work assume the fixed orientation while evaluating different classification algorithms. The experiment is carried out with addition of fourth dimension called magnitude of a sensor.

The processed data is analysed using machine learning tool called WEKA (Waikato Environment for Knowledge Analysis). The pre-processed was converted to ARFF (Attribute-Relation File Format), which is suitable for WEKA. They used 10-fold stratified cross validation technique to evaluate different classifiers. The WEKA is used to repeat different experiments as it enables to repeat different experiments multiple times.

2.1.2 Performance Analysis

The evaluation of accelerometer and gyroscope is done on four body positions using seven commonly classifiers i.e. Naïve Bayes (NB), Support Vector Machines (SVM), Multilayer Perception (MLP), Least Mean Square (LSM), Instance based learning (IB1), J48 and Decision Trees (DT). The author used accuracy or True Positive Rate (TPR) as performance metric. The TPR of a classifier means of the amount of correctly classified examples of a specific class out of its all examples. The overall TPR of a classifier is the weighted (by class size) average of individual TPR for all classes being recognized. The results give an overview of how different classifiers behave in terms of their classification accuracies with a gyroscope, an accelerometer and their combination. Some of the results captured and discussed below

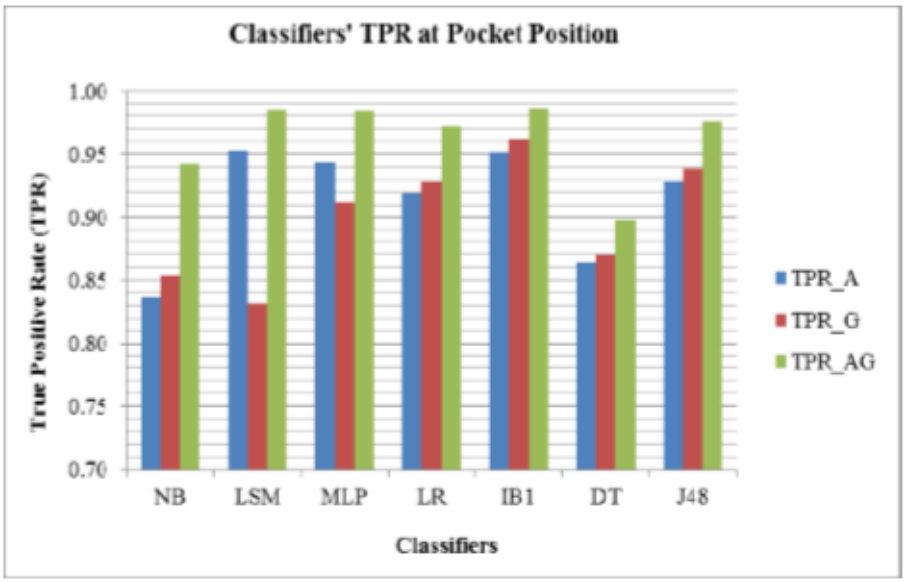


Figure 1 Pocket Position

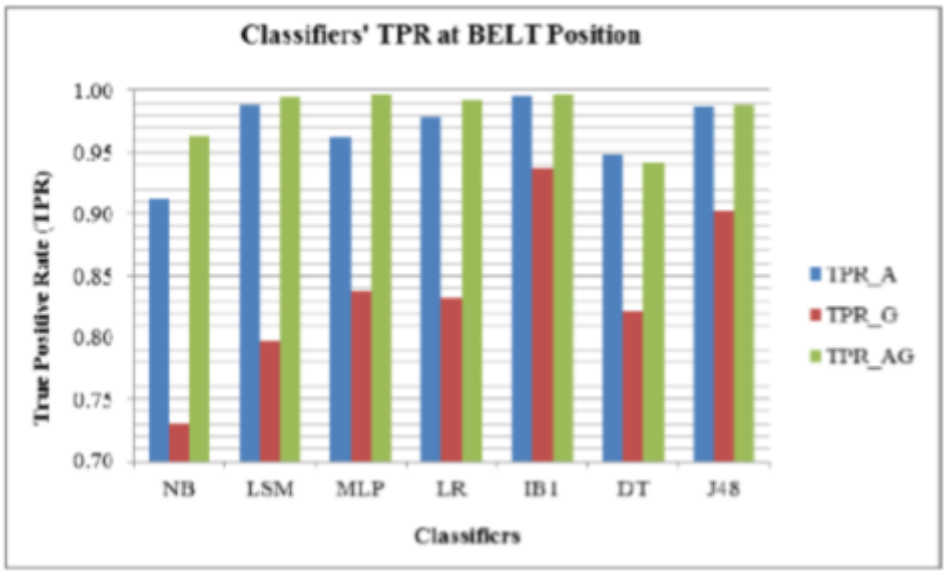


Figure 2 Belt Position

From the above Figure 1 and Figure 2 shows the overall accelerometer (TPR_A) performs better than gyroscope (TPR_G) in recognizing activities except pocket. TPR_AG gives the combination results of accelerometer and gyroscope the results showed improved than their improved results. In pocket position, the overall gyroscope performs slightly better than accelerometer. In the same way the magnetometer is analysed the results show the magnetometer performance is poor when compared with other sensors its results are shown in [3]. The combination of the gyroscope and the accelerometer performs better than their individual performances in all cases except for DT at belt and pocket position. In the paper

activities were analysed on four body positions but results for pocket position only shown. The analysis of the statistical data is presented in the form of plots. For the other positions the results are summarized due to limited space.

2.1.3 Conclusion

The evaluation of sensors is done using seven commonly used classifiers on four body positions (arm, pocket, body, wrist). The gyroscope not only improves the recognition performance but produces reasonable performance when used alone. The combination of accelerometer and gyroscope (TPR_AG) improves the TPR. Moreover, the feasibility of magnetometer in activity recognition can be studied further [3]. Based on evaluations, it is difficult to make exact generic statement about the role of these sensors in activity recognition process for all situations.

2.2 Tracking Motion Context of Railway Passengers by Fusion of Low-Power Sensors in Mobile Devices

In this paper Station Sense, a novel mobile sensing solution for stop-and-go patterns of railway passengers is proposed. The experiment is carried out in Japan to solve the poor tracking in modern trains. Station harnesses characteristic features in ambient magnetic fields in trains to find candidates of stationary periods, and subsequently filters false positive detections by a tailored acceleration fusion mechanism. In the experiments conducted StationSense can identify periods of train stops with 81% accuracy, which is almost 2 times higher than the existing accelerometer-based solutions.

2.2.1 Accelerometer-Based passenger tracking systems

StationSense effectively combines measurements from accelerometers and magnetometers in mobile devices by a tailored data fusion algorithm. A logger application is installed in Android device to record accelerometer and magnetometer readings at 60Hz sampling frequency. During the experiment phase each participant carries two logger phones: one in hand and other in trouser pocket. The participants record actual departure /arrival time by tapping buttons on a dedicated annotation app. The samples collected annotate sensor data for 18 separate train rides.

To detect stops at stations, SubwayPS [4] employs a hard threshold on magnitude of acceleration. It filters out the gravity components from the raw sensor readings and then calculates the magnitude of linear acceleration. It gives the vehicle in motion if average acceleration magnitude over a 2-second window exceeds a pre-defined threshold. Otherwise the vehicle is considered stationary.

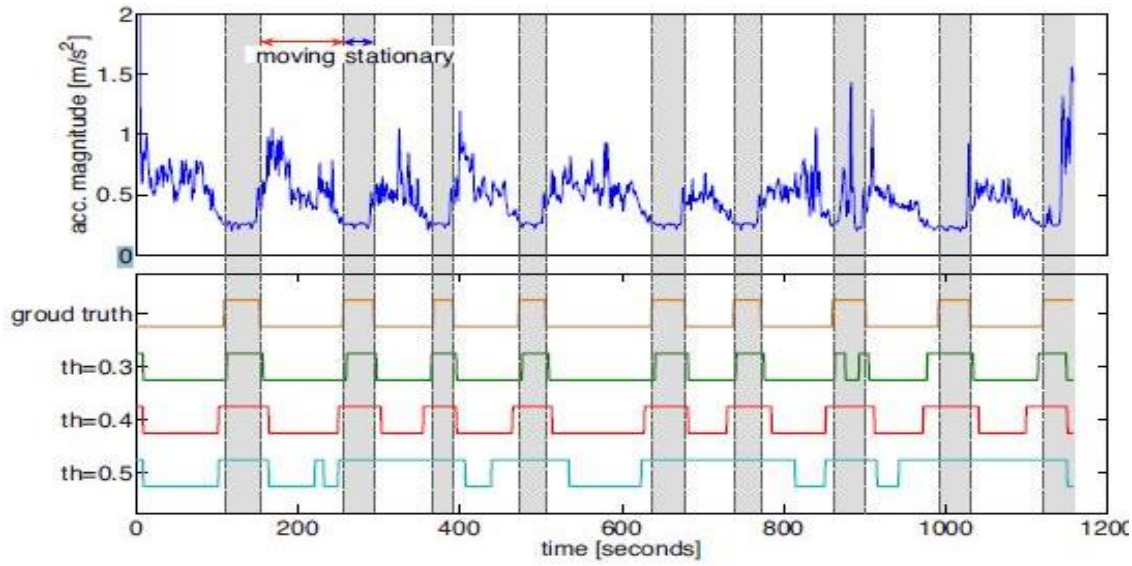


Figure 3 Train stop detection performance by SubwayPS

Figure 3 shows stop detection performance of SubwayPS. The upper plot shows time series acceleration, magnitude where gravity components are filtered out. The lower plot shows actual motion state and estimated state calculated by SubwayPS with acceleration thresholds of 0.3, 0.4, and 0.5 m/s^2. By setting threshold to 0.3 m/s^2, the system can accurately identify periods of time when the train is stationary. The smaller variations in the acceleration feature makes it difficult to accurately identify the time of state transitions.

2.2.2 StationSense Passenger Tracking System

In this it first analyses magnetometer readings to calculate time series probability that a train is stationary. The sensor measurements analysed gives the thorough analysis of magnetic signals observed in acceleration and deceleration periods. However, difference from stationary periods, resulting in potential false positive detections. These wrong detections are filtered by acceleration fusion and then search for optimal boundaries between adjacent moving/stationary periods by fusion of the stop probability and unique acceleration signature, which is typically observed when a train stops and restarts moving.

Analysing the dataset, gives the spike in magnitude of horizontal acceleration components exactly when trains stop at and depart from stations. Station-Sense reduces the solution space for optimal boundaries by finding such peaks.

2.2.3 Magnetic Field Sensing

The magnitude of magnetometer reading are considered along x,y and z axis. The magnitude varies over time significantly and doesn't have clear correlation with the train's motion state. The author chooses the window size based on typical parameter configurations (*i.e.*, 1-2 seconds) based on previous literatures and set the sliding margin to 1 second. In moving periods large variance values are frequently observed and it sticks to fairly low level for

almost the whole stationary periods. This suggests that variance in magnitude of the magnetic field serves as a powerful feature for classifying motion state of trains.

2.2.4 Conclusion

StationSense a novel mobile sensing solution for precisely tracking motion context of railway passengers. It harnesses characteristic feature in ambient magnetic fields. Through field experiments, StationSense can identify stationary periods of trains with accuracy of 81%, which is 80-93% higher than the existing accelerometer-based solutions.

2.3 By Train or By Car? Detecting the User's Motion Type through Smartphone Sensors Data

This paper discusses about the context-aware computing by focussing on a specific research problem like detecting the user's motion type from smartphone sensors data. They took the challenge of recognizing current user's moving *i.e*. by car or by train which involves the considerable efforts if such detection must be performed in an automatic way. To achieve these results, the problem is addressed into motion type recognition with three novel contributions. First, showing some experimental results gathered from smartphone sensors for different class of motions and demonstrate the existence of patterns. Second, propose a classification methodology to recognize these features by machine learning techniques. Finally, implement the proposed techniques on Android application, called What Am I Doing (WAID).

2.3.1 Data Classification

The process starts with collecting sensor data. For this purpose, an android application is developed to collect the sensor data (set to 10Hz) for three motion types such as walking, moving by car and moving by train. The sensor includes accelerometer and gyroscope. After collecting the data each data set is divided into consecutive non-overlapping time sequences length T (equal to 10 seconds in this tests). For each sequence the maximum, minimum, average and standard deviation are extracted. A set of features associated to accelerometer and gyroscope used for training set. Finally, a data set of accelerometer and gyroscope are generated and combined for the training set which is further used for classification.

In these three different classifications are considered i.e. Random Forest (RF), Support Vector Machine (SVM), Naïve Bayes (NB). In this RF was used to discover possible patterns over sub-sets. SVM are used to identify possible spatial relationships and NB for reflecting the stochastic nature of the environment where the training set was built. From the training phase Random Forest shows the highest performance (97.71%) which is further used for testing the samples. The accuracy of the classifier is further independent of the training rate. The latter result is confirmed by the application using high sampling rate.

2.3.2 The WAID (What am I doing) application

The WAID application includes a motion-type recognition algorithm based on Random-forest classifier (details ref [5]) and a context-profiler module to adopt smart-phone configuration to the current motion type detected [2]

Figure 4 WAID Interface

In the Figure 4 shows the WAID application user interface. At each time, the user can configure the profile associated to a specific motion. It runs into two different modes: training and predicting. In training mode, the application collects sensor data samples and gets information about the ongoing activity by the user's feedback and then stores the information. The application runs in the background as an Android Service and exports the motion type via an Android Content Provider. In predicting mode, the application works similarly but it determines the current motion based on the stored data.

2.3.3 Conclusion

This paper addresses the problem of determining the user's motion type from smartphone sensors data. This paper evaluates different feature extraction techniques and classification algorithms. A history-based recognition technique is used to reduce the occurrences of classification errors by exploiting the temporal correlation among consecutive sensor readings. The application is developed using Random Forest showing highest accuracy of 98%. Finally, integrated the motion type recognition algorithm into an Android application.

2.4 Estimating driving behaviour by a smartphone

In this paper a system is developed to estimate driving profile of drivers which can detect risky driving patterns likely to be generated by drowsiness, inattention or traffic violations. This ensures safe driving habits, decrease accident rates and provide the passenger a safe travel. This study conjugates the fuel efficient and better driving habits among its adopters. This paper gives information about theoretical background relating to end-point detection, wrapping algorithms, and Bayesian classification.

2.4.1 System Block Diagram

In the Figure 5 the first step is the data acquisition and pre-processing of the data via smoothing filter. Next, apply the endpoint detection algorithm to estimate the temporal range of the signal searching for important events. Once the events are detected we temporally scale the signal to identify the event within the selected portion of signal using DTW (dynamic time wrapping). The main reason behind DTW is to overcome different temporal durations of the same event across different drivers. After this initial step, the Bayesian classifier [6] is applied to identify risky of safe driving habit. Details of these steps are given in the following sections

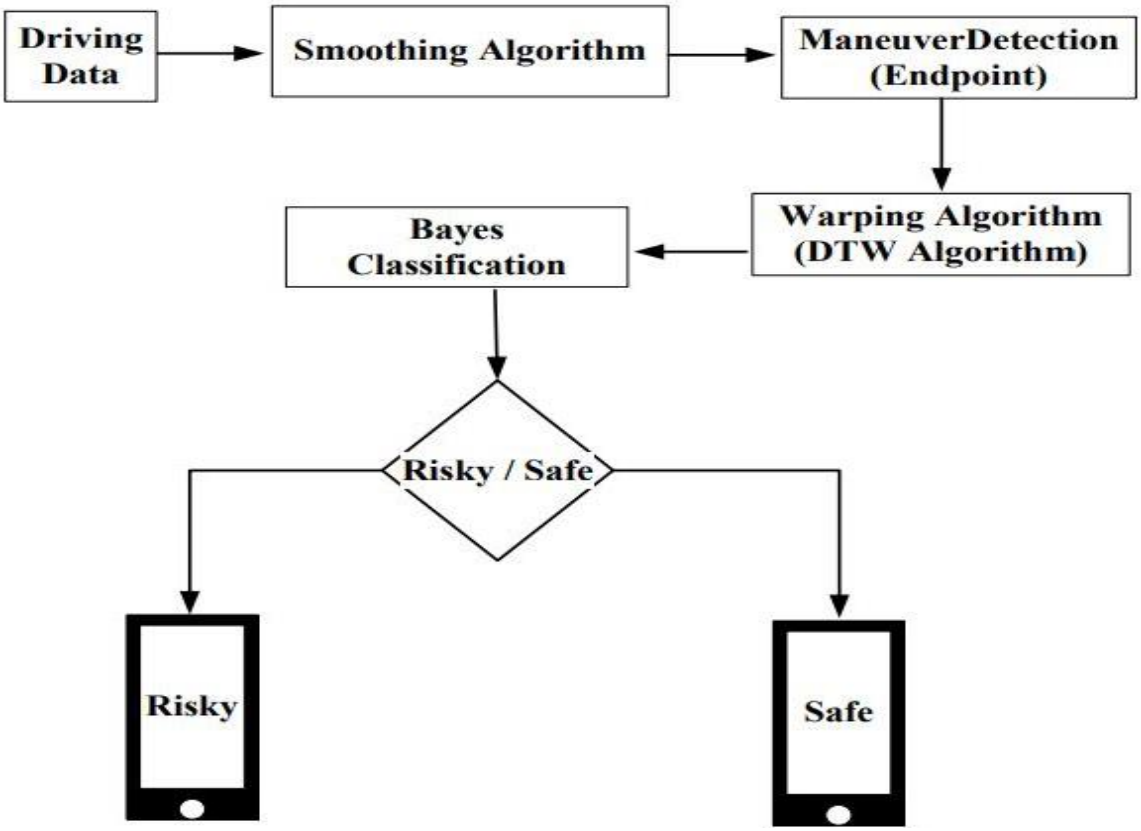


Figure 5 System Block Diagram

2.4.2 Endpoint Detection

In this project this is used to detect risky events caused due to sharp manoeuvres, unsafe turns and sudden braking. The reason for manoeuvres may be related to driver's present mood/behaviour such as aggressive driving, drowsiness. These signals are estimated with test data by end-point detection algorithm. The detection of left turn, right turn, lane departure acceleration is performed by matching training templates for these events. The algorithm works by organizing the samples using window method.

2.4.3 DTW Algorithm

This is used for optimal path estimation. In the data set the initial and final values are detected the system accommodates the difference between training and testing event durations based on the DTW Algorithm.

2.4.4 Bayesian Classification

Bayes classifier are family of simple probabilistic classifier. This based on applying strong independence assumptions between features. In this case safe and unsafe driving habits are considered. Given the prior probabilities the system output can be obtained by comparing the posterior probabilities.

2.4.5 Experiment Design

In this, they have chosen to analyse driving patterns of a group of 15; five of them have been selected from experienced drivers, the other five have been from novice ones, and rest had been selected randomly. The vehicle reference frame for the car is defined as follows; the +y axis points in the hood direction, the +x axis points in the driver side direction and the +z axis points in the roof direction of the vehicle.

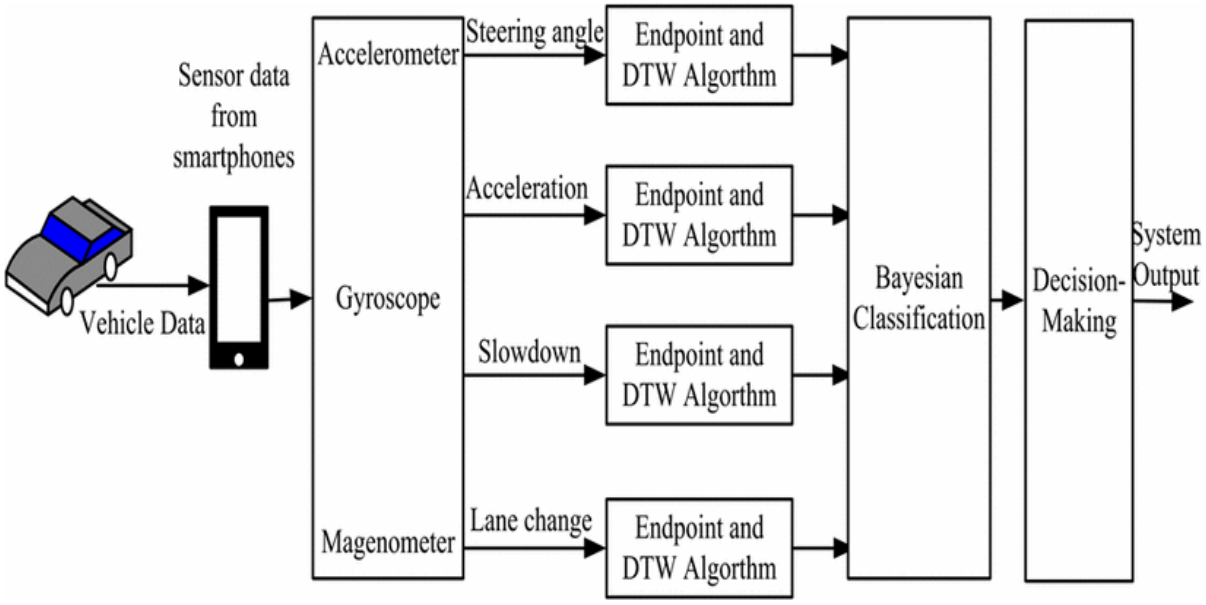


Figure 6 Experimental Setup

In the Figure 6 experimental setup is given. The accelerometer data is filtered by applying smoothening filter. The range of actions are defined from the endpoints of the events. Then a testing and training dataset is provided. The events are classified using Bayesian classifier and further used for algorithm development showing highest accuracy of 93.3%. The final output is recorded as safe or unsafe driving.

2.4.6 Results

In the experiment, a group of 15 driving patterns are analysed. Five of them are experienced, the other five are novice and the other five are random. The results of Bayes classification are shown in Figure 7. The final output of the classification is recorded as safe or unsafe driving.

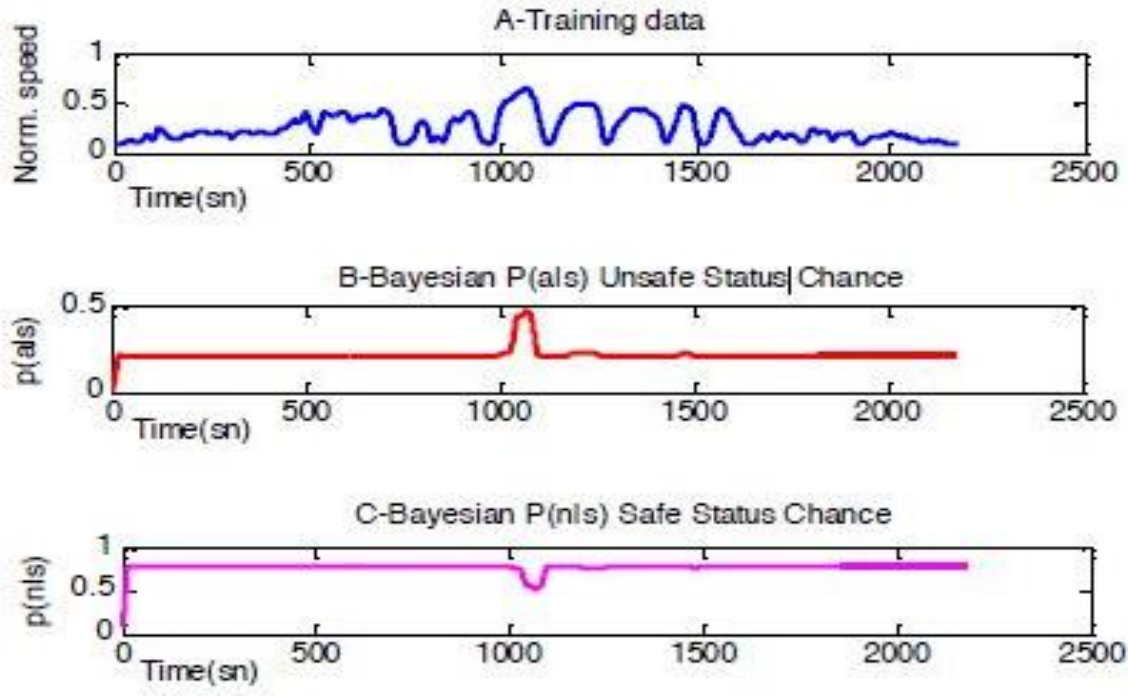


Figure 7 Bayesian classification using steering wheel

(a) Template data for the system
(b) P(a|s) unsafe probability obtained by the result of Bayesian classification process
(c) P(n|s) safe driving probability obtained by the result of Bayesian classification

2.4.7 Conclusion

This paper gives the investigated driver behaviour as safe or unsafe using optimal path detection algorithm and Bayesian classification [6] . On the other hand, both computational and implementation cost of the proposed method are promising in compared to the other methods.

2.5 SubwayPS: Towards Smartphone Positioning in Underground Public Transportation Systems

In this project, the problem of underground transport positioning on smartphones is introduced based on accelerometer positioning technique that allows smartphones to determine location substantially better than baseline approach. The project highlights several immediate applications of positioning in subway networks and present MetroNavigator.

2.5.1 Introduction

A smartphone's ability to detect its location-based services to crowdsourcing. Underground public transportation systems, which are largely inaccessible to GPS signals and often out of network range. As a result, the users cannot take the advantage of popular location-aware applications.

In this paper SubwayPS presents, the first positioning technique that allows modern smartphones to determine their location while travelling in an underground public transport. This project uses accelerometer and gyroscope as an input origin location and end destination. The technique Implemented can make mobile maps useful in underground transportation networks.

2.5.2 The SubwayPS Technique

This technique shows to achieve reasonable positioning accuracy in underground public transportation network. This also shows how accuracy can be improved by fine-tuning. SubwayPS uses accelerometer and gyroscope by combining the accelerometer measurements on all three axes. The gyroscope is used to filter the gravity factor out of the accelerometer measurements. As such accelerometer reads initially $0m/s^2$ in all axes. The axes are not swapped when the device's screen orientation changes. Collecting data is done in small periods when there was generally less acceleration on all axes of the accelerometer. This is because of multiple small accelerations in all directions. Figure 8 shows the data collected stations are clearly visible in the data in that they have accelerations closer to zero and less variation. A general acceleration value is calculated based on three-axis inputs. The blue dotted lines indicate time in minutes while red intervals indicate stations [4].

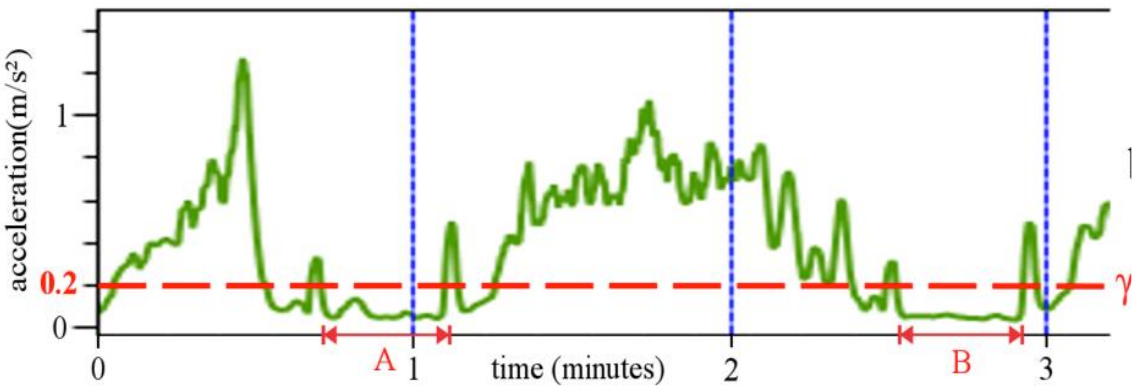


Figure 8 Visualization of accelerometer values

The parameters considered here are:

n = *window size of moving average*

γ = *threshold*

δ_{below} = min. amount of samples below threshold

δ_{above} = min. amount of samples above threshold

The values for each parameter was determined empirically using data from four different subway systems. SubwayPS requires at least δ_{below} samples below the threshold before it marks the current accelerometer measurements as a stop, while at least δ_{above} samples above the threshold are required for SubwayPS to conclude that the train is moving. The values are used to prevent flagging user movement as train movement. The parameters specifically tuned to the data collected on each subway system.

2.5.3 Implementation

The SubwayPS was implemented on the Android platform as MetroNavigator application on both smartphone and smartwatch versions. This uses virtual linear accelerometer provided by Android operating system. The accelerometer readings are interpreted by the SubwayPS technique, which is implemented as a background service [4]. MetroNavigator is a proof of concept application built to use SubwayPS as its positioning technology. It consists of miniature map and event cards.

2.5.4 Evaluation

To evaluate SubwayPS, it is studied in three ways. They are as below

Study 1:

The goal was to collect accelerometer data from a variety of diverse subway systems and use this data to inform and evaluate SubwayPS.

Study 2:

A second user study was conducted with the MetroNavigator application to directly compare the use of SubwayPS against an approach that merely interpolated from an official timetable.

Study 3:

To test the feasibility and appeal of SubwayPS and its proof-of-concept implementation in MetroNavigator, a user study was conducted. The study was conducted across two days.

2.5.5 Results

The average number of stations per track was 9.23 and the average track length was 21.51 minutes. The analyzation of data determines the value for each of each Subway parameters. In the experiment out of 103 out of 120 stops (105 stations and 15 “in-between” stops) were classified correctly (85.8% excluding the start stations). No false positives were recorded. These results are somewhat comparable to those of a study done by [7] using four tracks in the subway system. Across all 50 trips, 282 of 335 stops (300 stations and 35 “in-between” stops) were classified correctly (82.7%, excluding the start stations; 85.0% including the start stations).

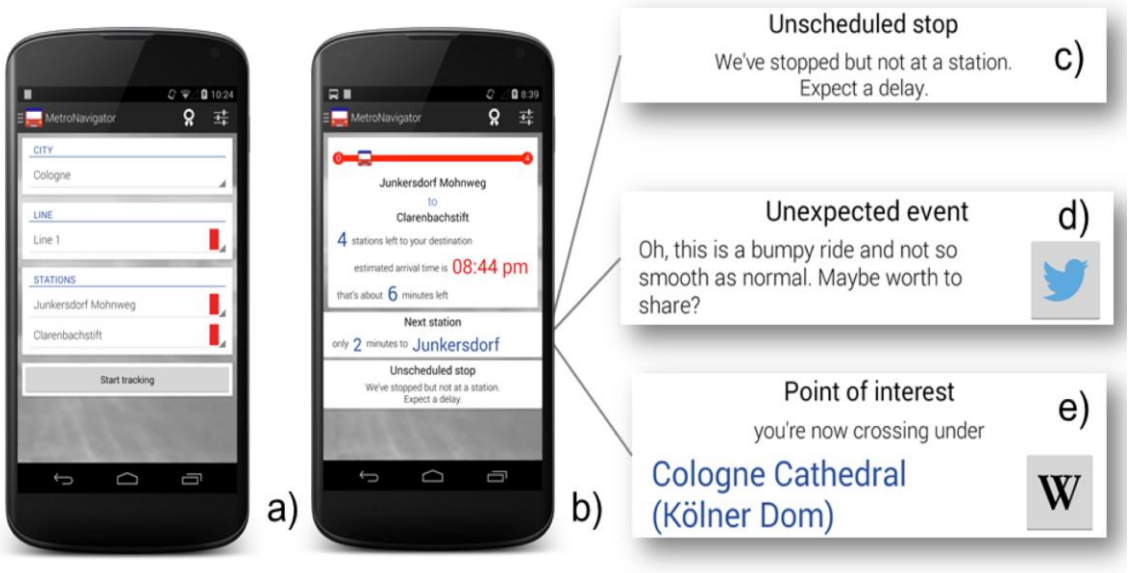


Figure 9 MetroNavigator application showing different event cards

The main advantage of MetroNavigator for most participants was the ability to be guided to their destination stop. It is shown in the above Figure 9.

2.5.6 Conclusion

In this paper, SubwayPS presents a smartphone positioning technique that puts users “back on the map” SubwayPS does not require any instrumentation of the environment. It can be easily on any smartphone containing accelerometer and gyroscope both of which are standard on many modern smartphones. The evaluation shows the accuracy can be increased if its parameters are tuned specifically. This tuning is simple, and merely requires a single user to collect accelerometer data as they travel on subway. In the short term, SubwayPS is straightforward enough to be implemented directly into a mobile app like using MetroNavigator.

Future work is proceeding along three directions. First, to implement on Wi-Fi based positioning correction to correct drift errors. Second to implement more sophisticated stop detection approaches using trained models. Finally, to consider the means by which crowdsourcing may be used to collect training data.

3 Neural Networks and Machine Learning

This topic presents neural network of large class with different architectures and machine learning process. The most useful neural networks in function approximation are Multi-Layer Perceptron (MLP) network. MLP consists of input layer, several hidden layers and output layer. When the network weights and biases are initialized, the network is ready for training. The multilayer feedforward network can be trained for function approximation or pattern recognition. The training process requires a set of examples of proper network behaviour. The following are some of the approaches used in the machine learning data processing.

3.1 Machine Learning

After obtaining the raw sensor data the log file is then processed onto Machine Learning techniques. Machine learning is a technology used to enable computers to analyse a set of data and learn from the insights gathered. It uses computational methods to learn information directly from data without relying on a predetermined equation as a model. By using complex algorithms an artificial neural network is simulated that enables machines to classify, interpret and understand data and then use the insights for solving problems or making predictions. Once a machine learning algorithm is programmed enriches itself based on the data fed.

Machine learning uses two types of techniques: supervised learning, which trains a model on known input and output data so that it can predict future outputs, and unsupervised learning for finding hidden patterns in input data. In the below Figure 10 shows the layout of the machine learning algorithm.

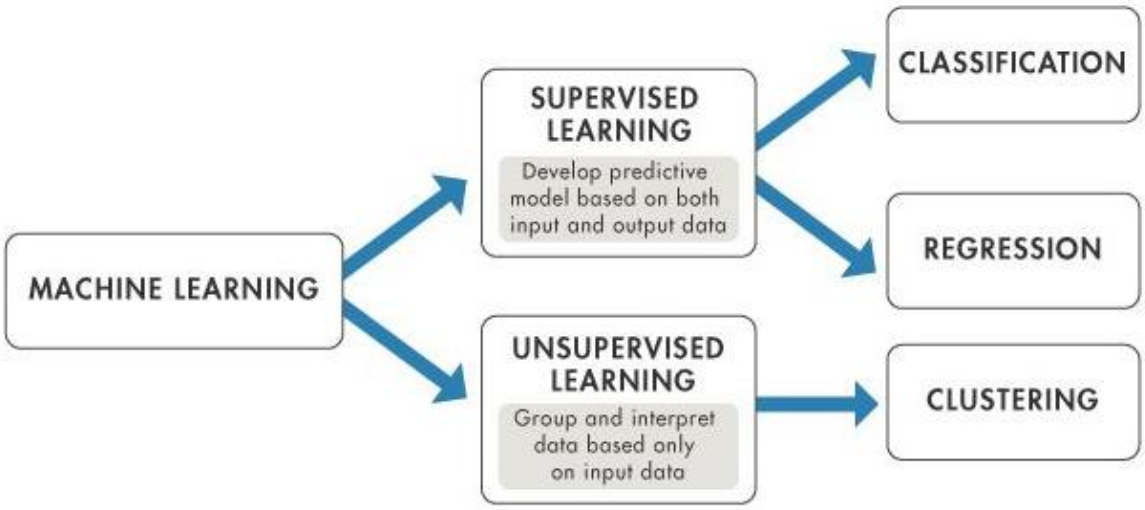


Figure 10 Machine learning layout algorithm

The machine learning is divided into two states supervised and unsupervised learning.

3.1.1 Supervised Learning

The aim of supervised learning is to build a model that makes predictions based on evidence in the presence of uncertainty. A supervised learning algorithm imports a set of known data and gives the output response of the data. The data set is carried for training phase and generates a reasonable prediction for response to new dataset. Supervised Learning uses classification and regression techniques to develop predictive models.

- Classification techniques predict categorical responses. These models classify input data into categories. Typically, applications include medical imaging, image and speech recognition, and credit scoring.

- Regression techniques predict continuous responses. Typical applications include electricity load forecasting and algorithmic trading.

3.1.2 Unsupervised Learning

Unsupervised learning finds hidden patterns or intrinsic structures in data. It is used for drawing inferences from datasets consisting of input data without labelled responses. Unsupervised Learning uses clustering technique.

- Clustering is the most common unsupervised learning technique. It is used for exploratory data analysis to find hidden patterns or groupings in data. Applications for cluster analysis are gene sequence analysis, market research, and object recognition.

3.2 Feature Generation

Feature generation consists of importing the large data sets into useful features providing relevant information that will have an essential role in the classification process. All the data gets imported, and then randomize the ordering. Classification problems require the selection of attributes to represent the pattern classification.

After feature transformation, it is possible that many of the features are either reduced or irrelevant are excluded without much loss of information and allowing to reduce the time consuming. So, keeping the dimensionality of the feature data as small as possible is important for fast data pre-processing [8]. Sometimes the data collected needs other forms of adjusting and manipulation. Things like de-duping, normalization, error correction, and more.

3.3 Classification

After feature extraction it is then carried on to Machine Learning Classification Learner. Choosing the right algorithm can seem overwhelming there are many available supervised and unsupervised machine learning algorithms, and each takes a different approach for learning. In this project supervised classification learning is used for developing predictive models. There is no best method or one size fits all. Finding the right algorithm is partly just trial and error. But algorithm selection also depends on the size and type of data used. Here are the some, the insights

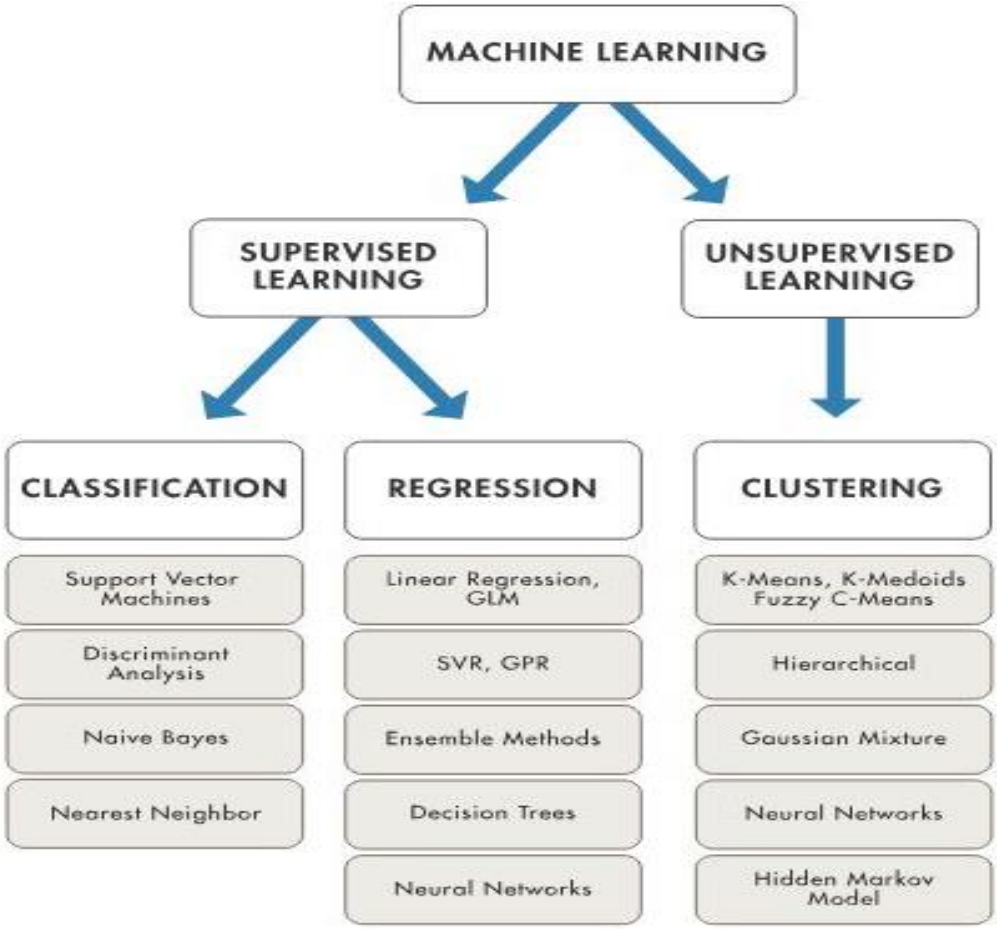


Figure 11 Machine learning techniques

The above Figure 11 gives the usage of machine learning techniques. Training a model with supervised learning gives prediction model. Choosing unsupervised learning gives the data exploration for training a model to find good internal representation, such a splitting data into clusters.

In this project the data stored in log file consists of numerical labelled data so supervised classification technique is used.

In a traditional activity recognition framework, the raw data set is usually fixed and prepared for processing. In this project the subject considered is metropolitan transport and should identify the instances of starting, moving and stopping of vehicle. As a consequence, the prediction accuracy changes when the system is used for different users. So, in this during the collection of data set the response is also taken parallelly. The challenge here is to efficiently process the data and predict the activity.

In this project the sample classifier, data training, classification is done using random forest, K nearest neighbour (KNN), Decision Tree and Support Vector Machine (SVM). These classification algorithms are discussed below.

3.3.1 Random Forest

Random Forest is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the classes output by individual trees. Random forest is a collection of trees, all slightly different. It is shown to work better as compared to other algorithms. It is an ensemble of decision trees such that each tree is grown independently using a randomly selected dataset while the distribution remains same. It is a meta estimator that fits several classical decision trees on various sub-samples and use magnitude to improve the predictive accuracy and control over-fitting. Each node within the trees is split using a randomly selected set of features. This randomness increases robustness to the algorithm against noise. RF self-testing is like a cross-validation that is repeated possibly hundreds of times, each time with different random partitioning of the data.

Random Forest has the ability to combine various ensemble of trees, extract a single tree from a forest, add trees to an ensemble and of course includes the training of algorithm, predicting and plotting the results. Default values were used for various variables involved in the algorithm.

In classification, the Random Forest algorithm employs a set of classification trees, while each tree is built on a bootstrapped sample of the original data. The classification trees are built based on recursive binary splits. In each tree, the best splits are determined using the Gini index which is the mean absolute difference is the average absolute difference of all data sets and the relative mean absolute difference is the mean absolute difference divided by the average, to normalize for scale.

This algorithm is used for feature selection. This is done by measuring the mean decrease of accuracy when a particular feature is removed from the set of features in the trees. The Random Forest algorithm assigns the importance score to each feature using the concept. The importance scores of the features in the random forest classifier can therefore be evaluated and used as a feature selection criterion.

3.3.2 Support Vector Machine (SVM)

The support vector machine (SVM) is a training algorithm for learning classification and regression rules from data, SVM can be used to learn polynomial and multi-layer perceptron (MLP) classifiers. SVMs are based on the structural risk minimisation principle, closely related to regularisation theory. This principle incorporates capacity control to prevent over-fitting. They are used in resolving problems as mentioned in [9] involving small samples, nonlinearity and classification of high-dimensional patterns.

When data are not labelled, supervised learning is not possible which requires unsupervised learning. The data is mapped into groups. The support vector clustering [10] algorithm applies statics of support vectors, developed in support vector machines algorithm, to categorize unlabelled data. The sample in these modes are classified into mStart, mMove and mStop according to the threshold and timestamp. It is as shown in figure in Figure 12.

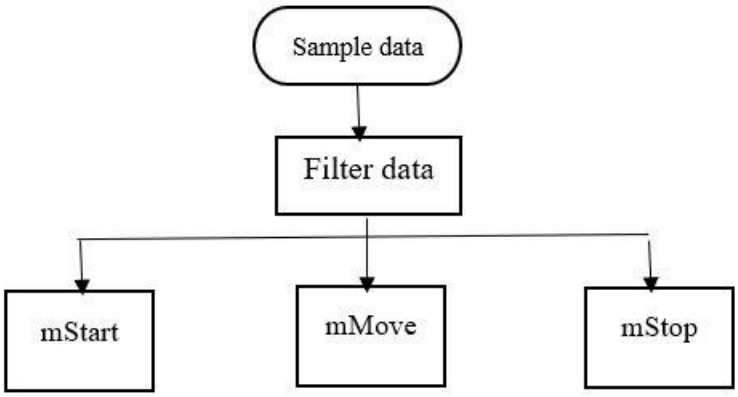


Figure 12 Sample data flow classification

A PCA (Principal Component Analysis) is proposed to be the best technique in multi variable analysis. The main purpose of this method is to reduce the dimensionality of the data set with many variables. The dimensionality reduction is performed by transforming data into a new set of variables containing the main components.

Considering a data set matrix X_{mxn} with n is the number of experiments, m is the number of samples. The first step of the PCA algorithm is to standardize data and expressed as following equation (1):

$$\mathbf{S} = \sqrt{\frac{\sum_{i=1}^{n}(X\mathrm{i}-\mathrm{X}^{-})}{(n-1)}} \qquad (1)$$

Secondly, calculating the covariance matrix from as follow in equation (2)

$$C^{mxn}=(c_{ij}, c_{ij}) \qquad (2)$$

where C^{mxn} is a matrix with m rows and n columns of the i^{th} dimension.

Thirdly, the eigenvectors and eigenvalues of the covariance matrix are calculated to collect the following characteristic polynomial equation (3):

$$|C-\lambda I| = 0 \qquad (3)$$

Where C is a square matrix, λ is called the eigenvalue of C. Thus, we can find eigenvalues which satisfy the following equation (4):

$$\det(|C-\lambda I|) = 0 \qquad (4)$$

Therefore, x is the eigenvector associated with the eigen value λ. The final step of the PCA is to choose components and to form a feature vector. The eigenvector with the highest eigenvalue is the principle component of the data set. In practice, there are many methods of identification and classification such as: Bayesian network etc. In this study, the identification method based on the SVM algorithm has been applied to analyse the accelerometer data.

3.3.3 K-Nearest Neighbour (KNN)

K-NN is a type of instance-based learning where the function is only approximated locally, and computation is deferred until classification. K-NN is used for classification or regression predictive problems. In this project it is used for classification of new object based on attributes and training samples. In k-NN classification, the output is a class membership. An object is classified by majority vote of its neighbours, from the most common class object. The neighbours are taken from a set of objects for which the class is known. This comes to picture in the training set for the algorithm when there is no explicit training step further. The better understanding of K-NN is explained in an example in [11].

The traditional K-NN text classification algorithm has three limitations: (a) calculation complexity due to the usage of all the training samples for classification, (b) the performance is solely dependent on the training set, and (c) there is no weight difference between samples. generally, larger values of k reduce the effect of noise on the classification but make boundaries between classes less distinct. A good k can be selected by various heuristic techniques. The classification rules are generated by the training samples themselves without any additional data. The K-NN is one of the most important non-parameter algorithms and it is a supervised learning algorithm.

The accuracy of K-NN algorithm can be degraded severely by the presence of noisy or irrelevant features or if the feature scales are not consistent. Much hard work is done on selecting features to improve classification. An approach is the use of evolutionary algorithms (heuristic approach) to optimize feature scaling. Another approach is to scale features by the mutual information of the training data with training classes. In binary classification problems, it is helpful to choose k to be an odd number. In similar activity recognition this algorithm works in combination with other helping methods.

The training examples are vectors in multidimensional feature space, each with a class label. The training phase of algorithm consists of storing the feature vectors and class label of training samples. In the classification phase, k is a user-defined constant and an unlabelled vector is classified by assigning the label which is most frequent among the k training samples nearest tot that query point. Usually Euclidean distance is used as the distance metric; however, this is only applicable for regression process. In cases of text is used classification is done based on another metric such as overlap metric or Hamming distance.

3.3.4 Naïve Bayees Classification

Naïve Bayes classifiers are a family of probabilistic classifiers based on applying Bayes theorem. Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (parameters) in a learning problem. It is a simple technique for constructing classifiers.

Bayes Theorem: Bayes Theorem works on conditional probability. Conditional probability is the probability that something will happen, given that something else has already occurred. Using the conditional probability, the probability of events is calculated based on prior knowledge.

The conditional probability is given as equation (5)

$$P(H\ |E) = \frac{P(E\ |H) * P(H)}{P(E)} \quad (5)$$

P(H) is the probability of hypothesis H being true

P(E) is the probability of the evidence

P(H|E) is the probability of the hypothesis given that the evidence is there

P(E|H) is the probability of the evidence given that hypothesis is true

Naïve Bayes Classifier: Naïve Bayes classifier uses Bayes Theorem. It predicts membership probabilities for each class such as the probability that given record or data belongs to a particular class. The class with the highest probability is considered as the most likely class.

Naïve Bayes classifier assumes that all the features are unrelated to each other. Presence or absence of a feature does not influence the presence or absence of any other feature. In real datasets, we test a hypothesis given multiple evidence. So, calculations become complicated.

To simplify the work, the feature independence approach is used to 'uncouple' multiple evidence and treat each as an independent one.

3.3.5 Decision Tree

A decision tree is drawn upside down with its root at the top. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In the Figure 13 on the left with bold text represents a condition based on which tree splits into branches. The end of branch that doesn't split anymore is the decision. Ignoring the simplicity of the algorithm the feature importance is clear and relations can be viewed easily.

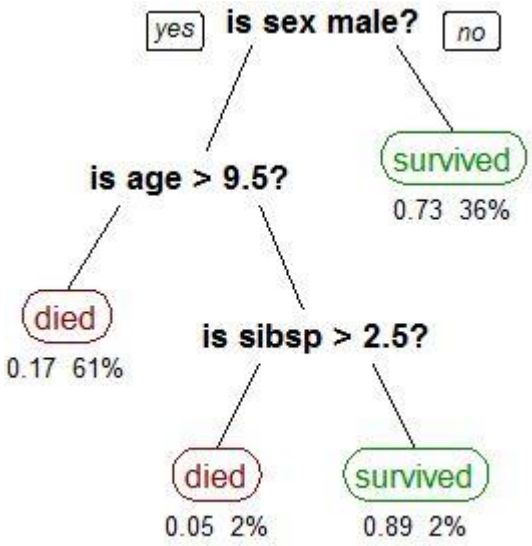


Figure 13 Decision tree

3.4 The Training Model

Classification is an important step in the prediction models. The most commonly used classifiers are support vector machine (SVM), K-nearest neighbour (KNN) and Naïve Bayes. In the process, a classifier needs to be trained by using labelled datasets (training data). There are two ways of training approaches: the offline and online mode. In the online mode, the training is performed on smartphone devices and the offline training is done in advance, performing on a local machine. Most of exciting studies use the offline training method [12] . The reason is to reduce the effort of computational costs. This project performs the training model in offline mode. The training process involves initializing some random values for mStart, mMove and mStop attempting to predict the output with those values. In the initial training the model's predictions are poor but can be improved by comparing the output parameters such that accuracy can be improved.

3.5 Evaluation

From the transformed data, training and testing set is selected. An algorithm will be trained on the training dataset and will be evaluated against the test set. This may be random splitting of data (into 60% for training, 40% for testing) or may involve more complicated sampling methods.

3.6 Cross Validation

A more sophisticated approach than using a test and train dataset is to use the entire transformed dataset to train and test a given algorithm. A method used in testing harness is

called cross validation. It first involves separating the dataset into several equally sized groups of instances (known as folds). The model is then trained on all fold's exception one that was left out and the prepared model is tested on that left out fold. The process is repeated so that each fold gets an opportunity at being left out and acting as the dataset. Finally, the performance measures are averaged across all folds to estimate the capability of the algorithm on the problem.

3.7 Prediction

In Prediction process a machine learning model is built. This model is used for testing by giving it a bunch of various features and where the value of machine learning is realized.

3.8 Conclusion

This project follows the above described popular neural networks and machine learning approach. Choosing a right neural network and apply to the data is heuristic. So here various neural network is applied, the best one is chosen based on the accuracy and minimum amount of time. Machine learning is a field that is continuously being innovated, it is important to keep in mind that algorithms, methods, and approaches will continue to change.

4 Machine Learning Tool

This section pertains the data mining tool, classification models used in developing the algorithm. The various types of diagrams (use cases, sequence) are included in this section each gives a process of the classification model. The classification model is obtained by using the machine learning tools like MATLAB, WEKA and Python the results are discussed in the following chapter 6.

4.1 Introduction

In this project MATLAB R2017A, python and WEKA are used as a platform to analyse the data processing. The reason why I considered this tool are MATLAB helps in giving a clear idea of the data plotted and the training phase, testing is very easy to perform. But finally, an algorithm should be exported to develop an application for android/ios application. The exported code generated in MATLAB doesn't support functions like fitcknn, fitctree. In the reference [13] gives the list of supported MATLAB functions for exporting the model. On the hand WEKA (Waikato Environment for Knowledge Analysis) is also a machine learning tool used for data analysis. This software provides the library supported files to develop an algorithm in java. The further discussion about WEKA is discussed below. On the other hand, recommendations from online forums [14] for simplicity coding and Ubirider work environment the entire data model is built in in python environment. The three functioning tools used in this project are discussed below

4.2 MATLAB

Machine Learning is a data analytics technique that teaches computers to do what comes naturally to humans learn from experience. MATLAB makes machine learning easy. With tools and functions for handling big data, as well as apps to make machine learning easy. MATLAB is an ideal environment for applying machine learning for data analytics.

MATLAB helps in:

- Compare approaches such as logistic regression, classification trees, support vector machines, and deep learning.
- Use model refinement and reduction techniques to create an accurate model that best captures the predictive power of data.

Machine Learning uses two types of techniques: supervised and unsupervised learning as mentioned in chapter 5. In this project supervised machine learning is used and classification process is selected. The process is described below

4.2.1 Classification

Classification is a supervised machine learning in which an algorithm "learns" to classify new observations from examples of labelled data. To explore classification models in MATLAB classification Learner App is used. For greater flexibility, the featured data is passed with corresponding responses to an algorithm-fitting function in the command-line interface.

4.2.2 Train Classification Models in Classification Learner App

Classification Learner App trains model to classify data. The app is used to explore supervised machine learning using various classifiers. After data exploration features are selected, specify validation scheme, train the model and analyse the result. The training process is automatic to search for best classification model type which includes decision trees, support vector machines, nearest neighbours etc. A dataset is supplied as an input data (dataset in log file) with known responses(labels) to the data. The data is used to train a model that generates prediction for the response new data. To use the trained model with new data, a model is exported to workspace and MATLAB code is generated to recreate the trained model. It is represented in Figure 14 below.

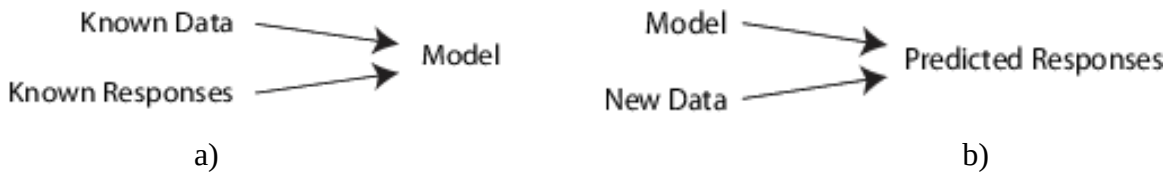


Figure 14 Response for classification model shown in a) response for prediction model shown in b)

The Classification process is carried out in three ways [15]:

A. Automated Classifier Training

B. Manual Classifier Training

C. Parallel Classifier Training

A. Automated Classifier Training

From the New Session data file is selected from the workspace or from the file. Specify a response variable and variables to use as predictors. On the Classification Learner tab, in the Model Type section, go to All Quick to train. This option will train all the model pre-sets available which are fast to fit. Click train. The Accuracy of each model appears, and the best accuracy is highlighted as shown in Figure 15. Click models in the history list to explore results in plots, confusion matrix, ROC curve.

Figure 15 Model accuracy

B. Manual Classifier Training

To explore the individual model types or a classifier one at a time manual classifier is used. In the Classification Learner tab, in the Model Type classifier type is selected. The Model Type expands the list classifiers with gallery presenting starting points with different settings, suitable for a range of different classification problems. Click Train after selecting the classifier. The characteristic of each classifier is shown in [16].

C. Parallel Classifier Training

Parallel Classification is supported by Parallel Computing Toolbox. During the training the app automatically starts a parallel pool. Parallel training allows to train multiple classifiers at once and continue working. When classifiers are training in parallel, the results can be examined from the model plots and initiate training of more classifiers.

4.2.3 Supervised Learning Workflow and Algorithms

The aim of supervised, machine learning is to build a model that makes predictions based on uncertainty of data. As adaptive algorithms identify patterns in data, a computer "learns" from the observations. When exposed to more observations, the computer improves its predictive performance.

The steps for Supervised Learning are

- Prepare data
- Choose an algorithm
- Fit a Model
- Choose a Validation Method
- Examine Fit and Update
- Use Fitted Model for Predictions

4.2.3.1 Prepare Data

Initially the process starts with an input data matrix from a log file. Each row represents one observation (accX, accY, accZ). Each column represents one variable predictor (mStart, mMove, mStop). The missing entries are represented with NaN values. Statistics and Machine Learning Toolbox supervised learning handles NaN values, either by ignoring them or ignoring any row with a NaN value.

4.2.3.2 Choose an Algorithm

Choosing an algorithm is based on several characteristics such as:

- Speed of training
- Memory usage
- Predictive accuracy on new data
- Transparency based on the reasons how an algorithm makes its predictions.

The selection of Classification Algorithms is studied from the Classification Algorithm characteristics [15] . The characteristics in any case may from the listed ones.

4.2.3.3 Fit a Model

The fitting of a model function depends on the type of algorithm used. The following are the list of algorithms and fitting functions used:

Table 1 Algorithms and fitting functions

Algorithm	**Fitting Function**
Classification Trees	fitctree
Regression Trees	fitrtree
Discriminant analysis (classification)	fitcdiscr
k-Nearest Neighbour (classification)	fitcknn
Support Vector Machines(classification)	fitcnb
SVM for regression	fitrsvm
Multiclass models for SVM	fitcecoc
Classification Ensembles	fitcensemble
Regression Ensembles	fitrensemble

Table 2 Algorithms and fitting functions

4.2.3.4 Choose a Validation Method

The model can be examined by using three methods:

- Examining resubstituting error
- Examining the cross-validation error
- Examining the out -of-bag error for bagged decision trees

Examining Resubstituting error: Resubstituting error is the difference between the response training data and the predictions the tree that makes response based on input training data. If resubstituting error is high, the predictions of the tree are not good. However, low submission error does not guarantee good predictions for new data. Resubmission error is often an overly optimistic estimate of the predictive error on new data.

Examining the cross-validation error: For better analysis of predictive accuracy rate for new data, cross validate the tree. By default, cross validation splits the training data into 10 parts at random. It trains 10 new trees, each one on nine parts of the data. It then examines the predictive accuracy of each new tree on the data not including in training that tree. This method gives a good estimate of the predictive accuracy of the resulting tree, because it tests the new trees on new data.

Examine Fit and Update

After validating the model, the next step is to opt for better accuracy, better speed. It can be done by:

- Change fitting parameters to try to get more accurate model
- Change fitting parameters to try to get a smaller model.
- Try a different algorithm

Use Fitted Model for Predictions

After fitting the model, the predictions for classification or regression fitted models use predict method in equation (6)

$$Y_{predicted} = \text{predict}\ (obj, X_{new}) \qquad (6)$$

4.2.4 Feature Selection

In the Classification Learner, identify the predictors that separate classes well by plotting different pairs of predictors on the scatter plot. The plot can help in investigate features to include or exclude. The scatter plot visualizes training data and misclassified points.

Before training a classifier, the scatter plot shows the data in Figure 16. If the classifier is trained, the scatter plot shows model prediction results in Figure 17. Choose features (mStart, mMove, mStop) to plot using X and Y lists under Predictors. Click Train to train a new model using the new predictor options. Observe the new model in the History list. The Current model pane displays how many predictors are excluded. The model can be improved by including different features in the model [22].

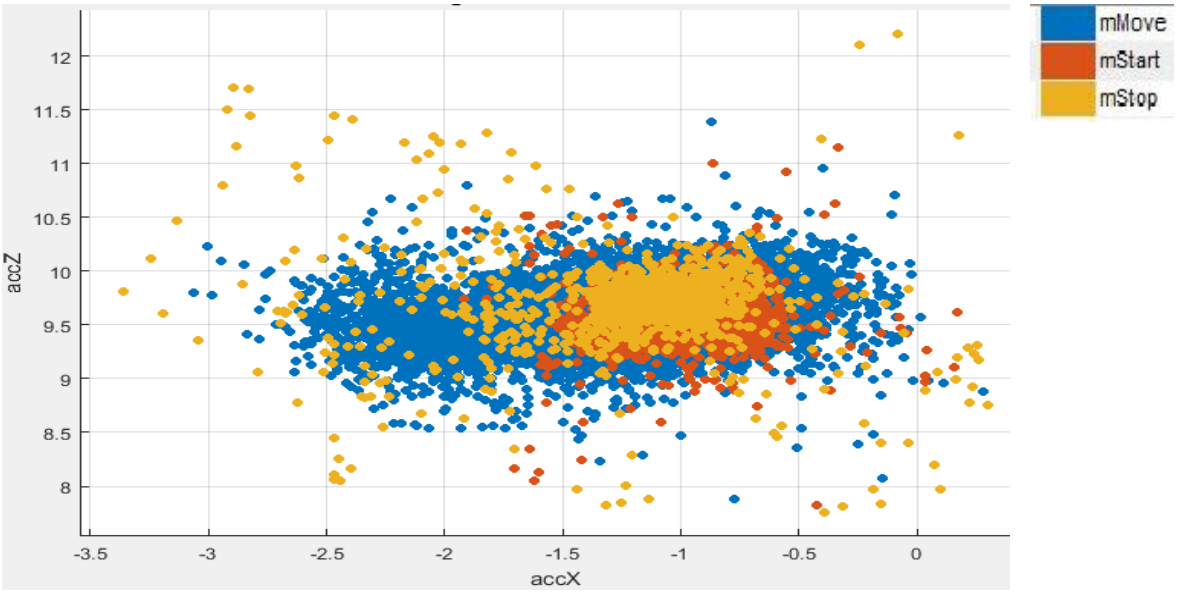


Figure 16 Scatter plot showing data before training

In the plot yellow indicates the data for mStart, blue for mMove and red for mStop. Figure 16 shows the scatter plot data before training. Figure 17 shows the scatter plot after training the data. After training the data scatter plot shows true classified and false classified data where false are indicated with cross mark and remaining indicate true classified.

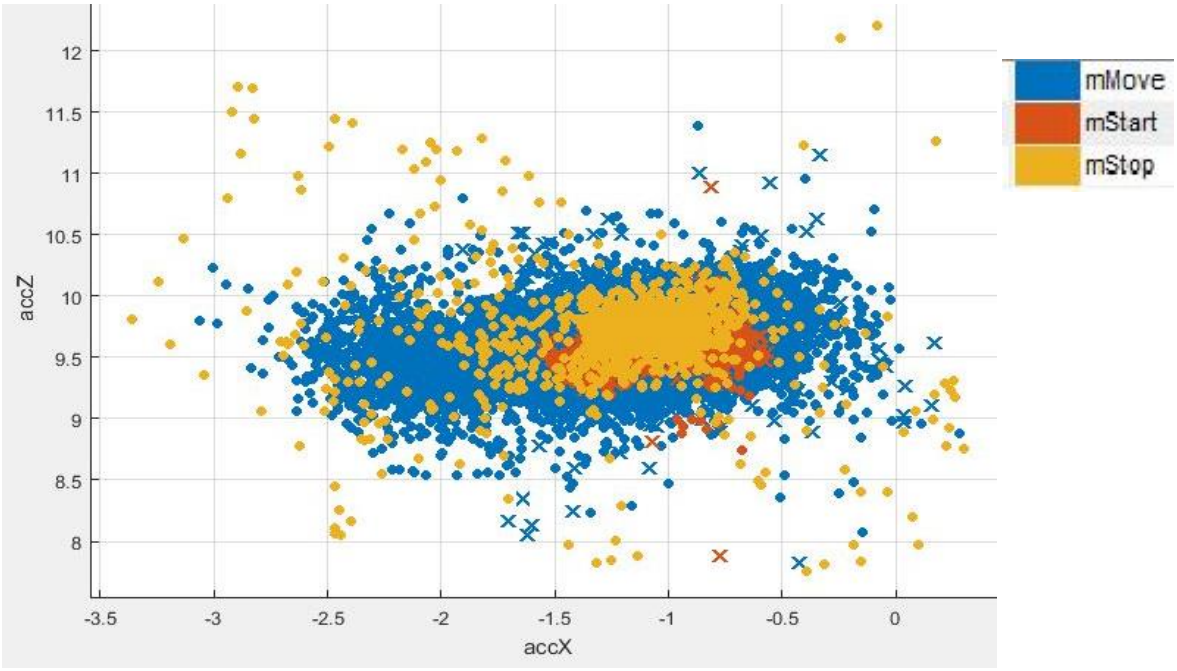


Figure 17 Scatter plot showing after training data

4.2.5 Compare and Improve Classification Models

After training the models select a model from the history list and explore the results in the plots. Compare model performance by inspecting results in scatter plot and confusion matrix. Select the best model in the history and try including and excluding different features in the model. Transform features with PCA (Principal Component Analysis) to reduce dimensionality. PCA is a technique used to emphasize variation and bring out strong patterns in dataset. If feature selection, PCA or new parameter settings improve the model accuracy, train the model with new settings.

Analysis

In MATLAB after training the data the performance of the data is studied by using scatter plot, cross validation matrix and ROC (Receiving Operating Characteristics) curve.

Scatter plot: The MATLAB functions plot and scatter produce scatter plots. It plots the discrete data identifying the relationship between two data samples. For example, coordinates [x,y]. The behaviour of scatter plot presents different groups (classes like start, move & stop) shown with the same symbol but different colours. After training the data plot displays the model predictions with true positive rates and false positive rates.

Confusion Matrix: Confusion matrix plots to understand how the selected classifier performed in each class. In the plot, the rows show the true class, and the columns show the predicted class. The diagonal cells show the true class and predicted class match. If the cells are in green, the classifier has performed well and classified observations of this true class correctly.

ROC Curve: It presents the selected classifier values with true positive and false positive rate. The curve shows a perfect result with no misclassified points is a right angle to top left of the plot. A poor result that is no better than random line at 45 degrees. The Area Under Curve (AUC) is a measure of the overall quality of the classifier. Larger AUC indicate better classifier performance.

Export Classification Model to Predict New Data

After classification model is created, export the best model to the workspace. To include the data used for training model select export model to the workspace as a structure containing classification object. After exporting the model, run the code generated from the app, a trained model is imported to workspace to predict new data. It is given as below equation (7)

$$Yfit = C.predictFcn(T) \qquad (7)$$

C ⟶ *name of variable*

T ⟶ new dataset

The validation Accuracy for new dataset is taken with input argument

$$[trainedModel, validationAccuracy] = trainClassifier(new\ dataset) \qquad (8)$$

4.3 WEKA

WEKA (Waikato Environment For Knowledge Analysis) is well-known machine learning software written in Java, developed at Waikato University in New Zealand. The application allows users a tool to identify hidden information database and file systems with simple user interface. The Weka workbench contains a collection of visualization tools and algorithms for solving real-world predictive analysis. The work flow is shown in the following Figure 18.

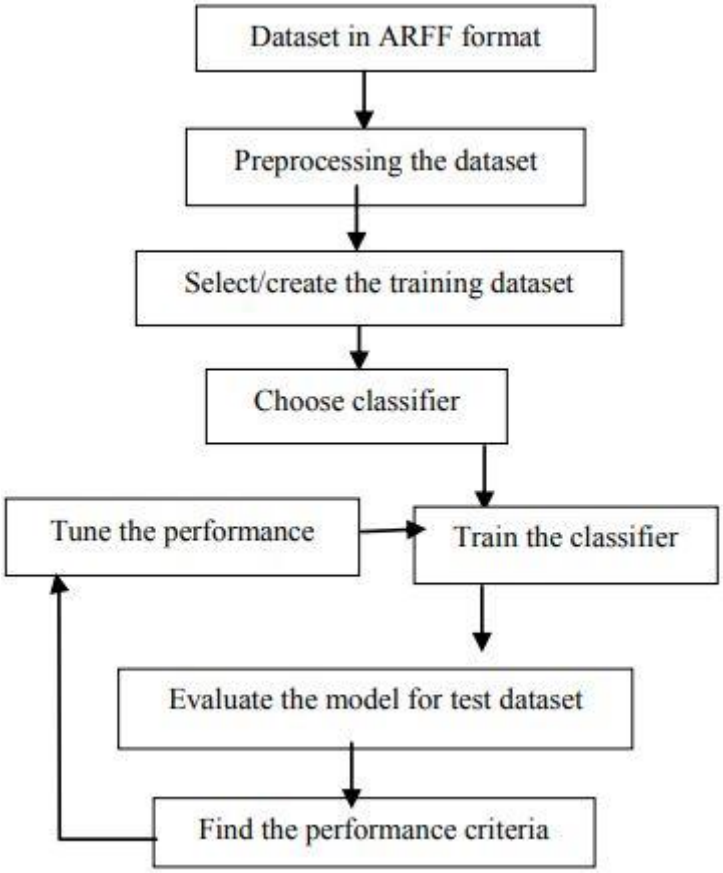


Figure 18 Data classification in WEKA

Classification is the process of finding a set of models that discuss and differentiate data idea, attributes and classes, for the purpose of being able to use the model to guess the class. Here in the project we have a dataset where each record has attributes for timestamp, acc_X, acc_Y and acc_Z upto K. The goal is to predict k for a given input record. In this Classification: K is a discrete attribute, called the class label whether Prediction: K is a continuous attribute Classification called supervised learning, because true labels (K-values) are known for data provided. The data is load and prepared in .arff format. This is pre-processing of data.

Classification process in WEKA is carried out as

1. Preparing the dataset.
2. Choosing the classifier and applying algorithm
3. Generate trees
4. Evaluate the model

1. Preparing the dataset: Preparing the dataset is about to load the dataset. In WEKA dataset can be loaded in csv format or arff. ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. After loading the dataset, it can be visualized as follow

In WEKA for successful examining of data Visualize panel gives the plots. Instances are shown as points with different colours for different attributes. A rectangle is swept out to focus the points on the dataset A classifier is also applied to visualize the errors by plotting the class against the predicted class. In Figure 19 shows the sample data visualization graph for individual coordinate axis (x,y,z) and activities (start, move, stop) for 14524 data samples (start= 1938, move=11387, stop=1199) . In the plot blue colour is for start, red is for move and sky blue is for stop. WEKA visualize also shows a scatter plot matrix where individual scatter plots can be selected and analysed further.

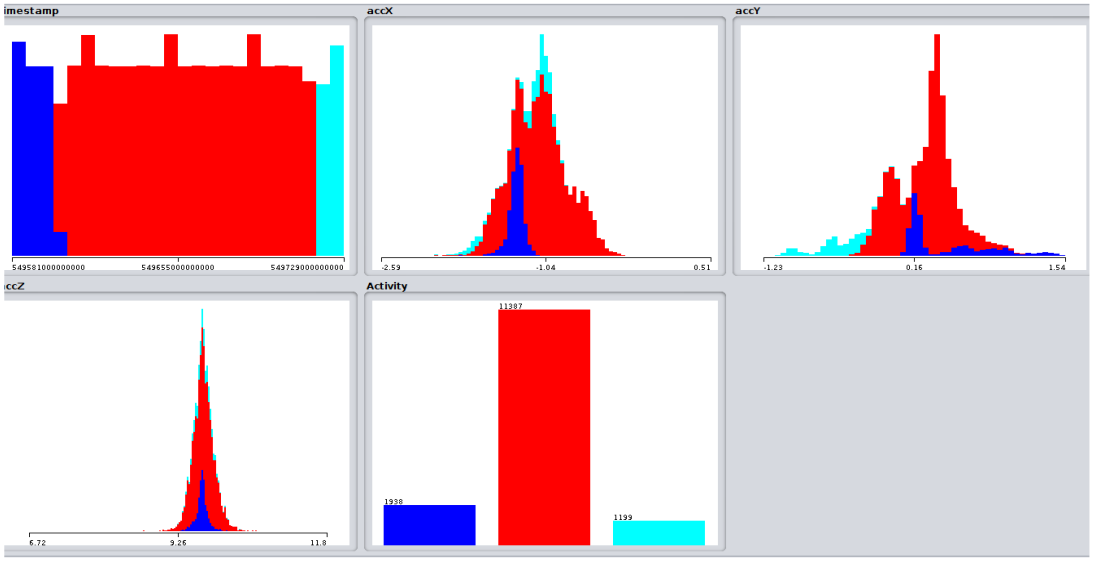


Figure 19 Sample data visualization graph

2. Choosing the classifier and applying algorithm

After pre-processing the data, the next step is to apply classifier. After choosing the classifier training and testing of data is done. It is carried out in four ways: Use training set; Supplied test set; Percentage split; Cross-Validation.

Use training set: This is highly optimistic [17] and not necessarily useful.

Supplied test set: It is an external file that can be supplied as a test set.

Percentage split: This splits the data into two sets of a certain percentage, one set for training and one for testing.

Cross-Validation: This is similar to percentage splits. It folds the data into 10 segments (for example) and repeat 10 (it is 10-fold): Use 9 folds for learning and leave 1-fold out for testing. This is the mostly used.

The above shown output results in Figure 20 uses 10-fold cross validation with K-NN classifier. It gives the output model with parameters like Area Under Curve (AUC), Kappa Statistic, Root Mean Square Error (RMSE) etc and several parameters with confusion matrix.

```
Correctly Classified Instances       23977               85.0248 %
Incorrectly Classified Instances      4223               14.9752 %
Kappa statistic                          0.5456
Mean absolute error                      0.0999
Root mean squared error                  0.3158
Relative absolute error                 42.546  %
Root relative squared error             91.358  %
Total Number of Instances            28200
```

Figure 20 Classification using KNN Algorithm

3. Generate Trees

Decision trees are a classic supervised learning algorithm. The main concept of decision tree is the following: starting the training data, build a predictive model which is mapped to a tree structure. The following is the sample data tree structure for various timestamp (54960000000000 in UTC format) for different classes with different time in Figure 21. The decision tree visualizes the data with respect to time for attributes. The data shown has a sequence from start to stop for an interval.

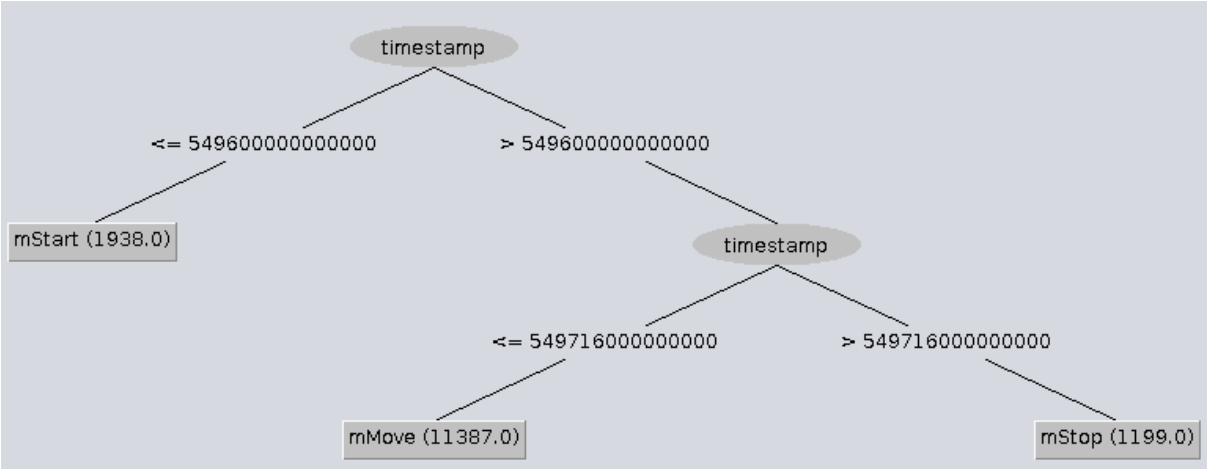


Figure 21 Sample data tree structure

In the Figure 21 visualizes the tree according to the timestamp. The data samples of mStart, mMove, mStop gets varied with respect to time it is shown by generating the tree,

4. Evaluate the result

Machine Learning algorithms have provided core functionality to many application domains such as computational linguistics, but it is difficult to detect accurate algorithm for implementing. The approach to validate is based on the accuracy of the problem and find the accurate algorithm [18].

In the test output cross validation is the most opted [19]. If a classifier learns and tests on the same data, a problem of overfitting is occurred. Therefore, the constructed model would exaggerate fluctuations in data, leading to a perfect accuracy on the test results. Applying a

new test dataset provides a poor predictive performance. So, to avoid overfitting, in all algorithms, is by using a k-fold Cross Validation method.

A k-fold cross-validation method, the original data is randomly portioned into k subsamples. Of the k subsamples, a single subsample is used as validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times, with each of k subsamples used exactly once as the validation data. The k results from the folds can be averaged to produce a single estimation.

The Detailed Accuracy by Class gives the performance of the predicted class. The following are parameters of the class shown in Figure 22 after classification:

- **TP Rate:** It's the rate of true positives (correctly classified instances) equivalent to recall.
- **FP Rate:** It's the rate of false positives (falsely classified instances).
- **Precision:** Proportion of instances that are truly of a class divided by the total instances classified as that class.
- **Recall:** Proportion of instances classified as a given class divided by the actual total in that class, equivalent to TP rate.
- **F-Measure:** A combined measure for precision and recall.
- **ROC (Receiving Operating Characteristics) area:** Illustrates the performance of the classifier, an "optimal" classifier will have ROC area values approaching 1.
- **MCC (Mathews Correlation Coefficient):** MCC has a range of -1 to 1 where -1 indicates a completely wrong binary classifier while 1 indicates a completely correct binary classifier.
- **PRC (Precision Recall):** The PRC area is calculated separately for class by treating instance of the class as "positive" instances and instances of all other classes as "negative" instances.

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0,076	0,027	0,317	0,076	0,122	0,827	mStart
	0,916	0,760	0,815	0,916	0,863	0,772	mMove
	0,537	0,048	0,470	0,537	0,502	0,922	mStop
Weighted Avg.	0,769	0,604	0,719	0,769	0,731	0,790	

```
=== Confusion Matrix ===

     a     b     c   <-- classified as
   151  1824    21 |     a = mStart
   326 10141   604 |     b = mMove
     0   478   555 |     c = mStop
```

Figure 22 Sample evaluation result

Statistical Test

To compare the performance of the classifiers with best results statistical test was used. The test consists of using a paired 10-fold cross-validation (Paired Corrected T-test) for each classifier, thereby obtaining the accuracy.

In the output result v/ * indicates statistically better (v)/ worse (*) for the baseline scheme with significant specified level.

The above are methods followed in analysing the data in WEKA. WEKA has a simulation tool and java library support to develop the algorithm for further implementing in mobile platform. WEKA has a large number of regression and classification tools. Native packages are the ones included in the executable Weka software.

4.4 PYTHON

Machine Learning in python is simple, elegant and math-like. It follows the regular machine learning sequence. The important thing here is the libraries used and their functionality. The imported libraries are:

- Numpy
- Pandas
- Matplotlib
- Scikit_Learn
- Scipy
- Statsmodel

Numpy: Numpy helps in arranging multi-dimensional array objects and tools for working these arrays efficiently. Numpy arrays can be sliced. Since arrays may be multidimensional, must specify a slice for each dimension of the array.

Pandas: Pandas is the library that helps to handle two-dimensional data tables in Python. In many senses it's really like SQL. With pandas, data is loaded into frames, select columns, filter for specific values, group by values, run functions (mean, magnitude, min, max, etc.), merge data frames. Pandas are imported for data cleaning, data handling and data discovery.

Matplotlib: The data visualization library used in Python is Matplotlib. It helps to understand the data and discover things from raw data format and communicate findings [20] efficiently.

Scikit-Learn: The library used for predictive analytics in machine learning is Scikit – Learn. It has several methods like regression, classification and clustering methods, as well as model validation and model selection.

Scipy: Scipy provides the core mathematical methods for complex machine learning processes in Scikit- Learn. The main reason for building the SciPy library is, it should work with Numpy arrays. Scipy provides efficient solutions to numerical routines like integration, interpolation, optimization.

Statsmodel: Statsmodels package provides complement to scipy for statistical computations including descriptive statistics, estimation and inference for statistical models [21].

While developing the project I started machine learning in MATLAB then I faced problems in exporting the code to java. Then I tried WEKA considered recommendation from [22] which has supporting library to develop in java, but the problem is cleaning of data and applying features like kurtosis, skew for time series data. On the other hand, python libraries provide simple import function and apply to the data. Below are the some of the features mentioned

4.5 Comparisons for Machine Learning tools used

MATLAB

Advantages

- Easy to import data clearing NaN (not a number)
- Easy to plot the raw data
- The complete machine learning analysis is done through classification learner app

Disadvantages

- MATLAB is proprietary. Tools used for machine learning are limited to organizations.
- Absence of functions like fitcknn, fitctree which are used for code generation
- Training and testing set should be of equal length.

WEKA

Advantages

- It is an open source can be integrated into other java packages.
- WEKA is best suited for mining association rules.

Disadvantages

- After building the model does not have the facility to save parameters for scaling to apply to future datasets.
- Does not have facility for parameter optimization of machine learning/statistical methods.
- Handling a bigger dataset gives the error “Out of Memory”.

PYTHON

Advantages

- It is an open source.
- Easy to plot the data make visualization using matplotlib.
- Reducing the code using lambda function for magnitude calculation.
- Plotting a sliding window and applying the features like kurtosis, skew.

Disadvantages

Python is server-side language rarely seen on client-side and implemented on smartphone-based applications.

4.6 Conclusion

When it comes to machine learning tool there exists abundance of tools. There exist different libraries at a time. Here it is used for data classification. Tools like MATLAB are proprietary, WEKA and python are open source. So, during this project these three tools are used based on the preferences [23].

5 Preparation of Experiment

The aim of the project is to develop an algorithm for the mobile application. The main feature of the application working is to guide the onboard users in underground, subways and places like non-existent GPS. So, the approach is to use the sensors available in the smartphones since modern smartphones are equipped with it and use less battery. By using the sensors (*i.e accelerometer*) collect the accelerometer incoming data and further process the data onto the machine learning and finally obtain algorithm based on the machine learning model. The layout of the project is shown in Figure 23.

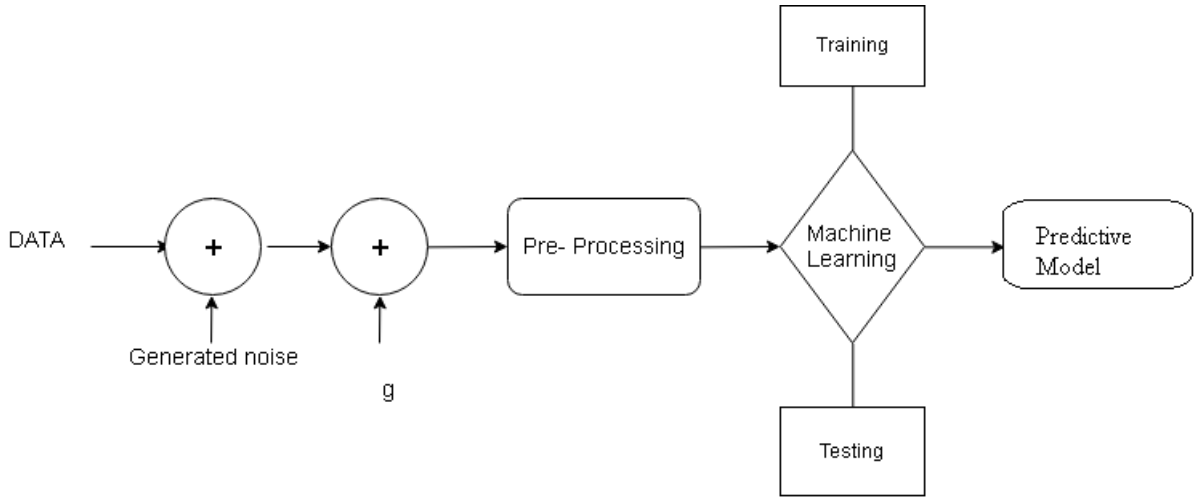


Figure 23 Project Layout

The major steps of transportation mode classification include: collecting data with smart phones, extracting features from the data, training the classifier, and classifying transportation modes using trained classifier. The collected data consists of generated noise (unwanted signal) with acceleration due to gravity (g) in one axis. The data is then transformed in pre-processing steps by extracting the features. The transformed data is applied onto machine learning process for classification process. Finally, a machine learning model is obtained.

5.1 Data Collection

At present smartphones are equipped with various sensors such as accelerometers, magnetometers and gyroscope to support user input and output. In addition to their main purpose the sensors can also be used to support positing, using inertial navigation techniques, when GPS or WiFi positioning are not available. Data from the accelerometer has the following attributes: timestamp, acceleration along x-axis, y-axis and z-axis. While collecting the metro activity data based on sensor input, a single sensor is not ideal; it may result in accuracy drop of 10% to 20% [9] compared to classification using multiple sensor input. Generally, two sensor types are desired: motion sensors (*i.e.,* Accelerometer and Gyroscope). In the transportation mode classification, motion sensors provide vibration or oscillation characteristics. So, in collecting the samples motion sensors has chosen as a base for the classification methods on the data from these sensors.

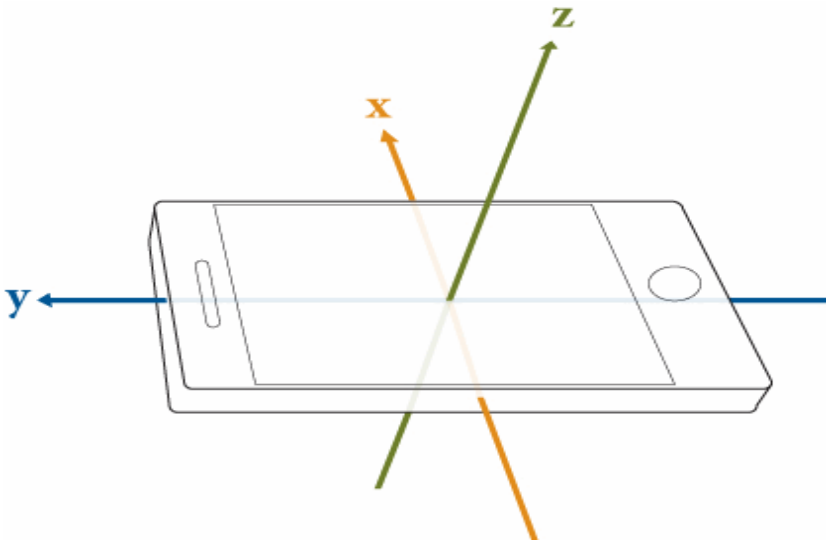


Figure 24 Coordinate system along a smart phone device

As mentioned in, the accelerometer is used to obtain vibration/oscillation features of user's bodies or their transportation mode and the gyroscope function is to extract the rotation feature. Importantly, not every device is installed equipped with gyroscope. So, we limit our motion sensing capacity to accelerometer. As shown in Figure 24, the device assumes a standard three-axis coordinate system to express data values.

This coordinate system is defined in relation to the device screen when the device is held in its default orientation. In this orientation, the *x* axis is horizontal and points to the right, the *y* axis is vertical and points up, and the *z* axis points outward from the screen facade. In this system, coordinates behind the screen have negative *z* acceleration values. While carrying out some physical activities or in pocket the device may be rotated and reversed relative to the user's body. Thus, we calculate the magnitude(M) of the acceleration vector from all the three axis for feature extraction in equation (8).

$$M=\sqrt{x^2+y^2+z^2} \quad (8)$$

The data is collected through Moto G4 plus device. When the data collection application runs, the incoming data is saved into a coma separated file (.csv) where each row consists of timestamp and accelerations along coordinate axis (x, y, z).

The Figure 24 shows the data plot for tri-axial accelerometer collected between two station intervals in Porto. The data is collected for metro activities like metro starting, moving and stopping irrespective of the human position. In project the activities are represented as mStart, mMove, mStop. When the metro doors are closed start button is in action the data gets collected for start period. The start period is from metro's stationary point to change in motion. Next follows the recording of metro moving for constant acceleration of motion and finally when the metro is approaching the station the speed is reduced and comes to stationary. During this period metro stopping acceleration is recorded. The data recorded for stop activity is when the metro reaches the station platform and speed of metro gets low. This is an approximate reference point considered to record the data. Typically, the mStart and mStop mode has short transmission state and it is difficulty to record accurately.

Here the important thing is the application runs with reference to the metro's opening and closing the doors. The data collection interval is considered between two stations. The application runs when the metro doors get closed record for start activity when change in speed the application is stopped and immediately started to record moving activity and metro approaches the destination station the speed gets low, the application is stopped and started immediately to collect data for stop activity. Finally, the data recorded gets when the metro reaches the destination station and doors are opened. This completes an interval for recording the data.

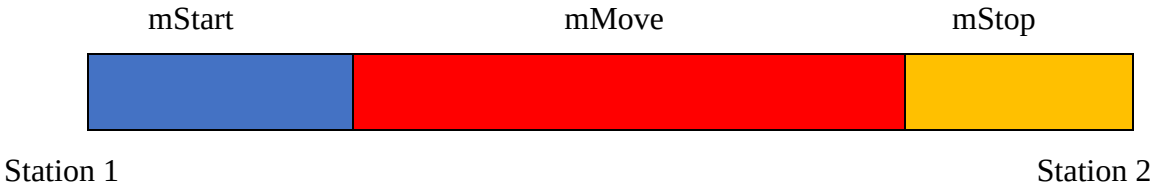


Figure 25 Data collection interval between two stations

During this process the data causes minute errors which doesn't affect much of the accuracy. The data is stored in a log file in .csv format. In the Figure 25 shows the activities (mStart, mMove, mStop) recorded for an interval between station1 and station 2.

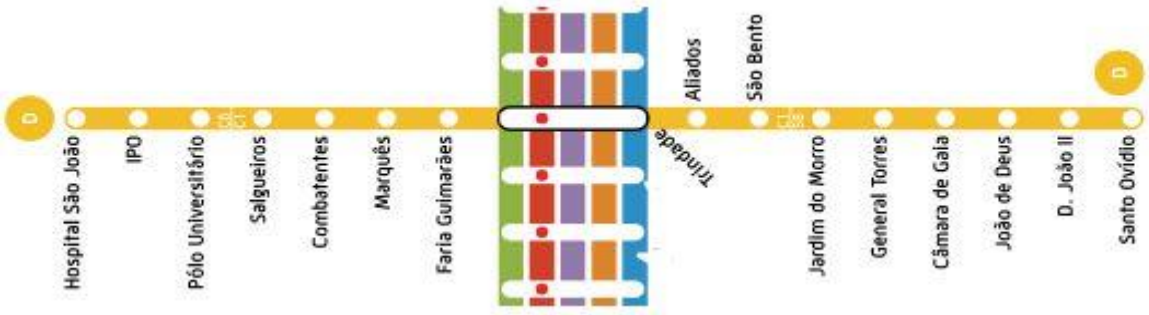
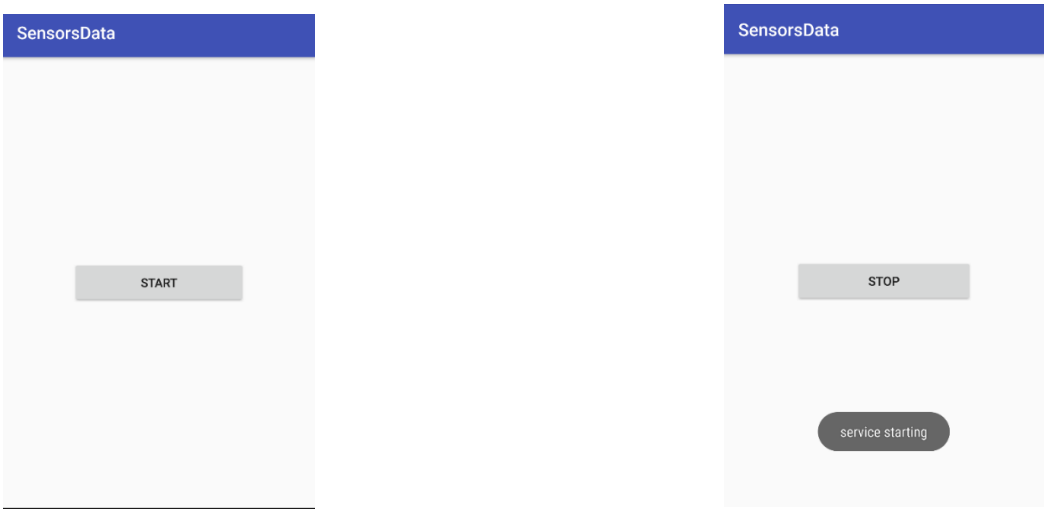


Figure 26 Data Collection Map Zone in Porto metro

In the Figure 26 shows the data collection zone in Porto metro line some of the stations are located in underground and some of them are on the ground. The data is collected from the stations on lying on the yellow line in both the cases. The stations like Hospital São João, IPO, Jardim do Morro, Geral Torres, Cãmara de Gaia, Joãa de Deus, D.João II, Santo Ovido are located on the ground and the stations like Pólo Unviversitário, Salgueiros, Combatentes, Marqués, Faria Guimaráes, Trindade, Aliados, Sao Bento lies underground. The travel times is 25 minutes and headway last for 5/10 minutes [24].The datasets collected consists of random data irrespective of ground position recording the metro acceleration with respect to metro start, move and stop activities.

5.1.1 Application image

In this section shows the images of the application used for collecting the raw data in Figure 27. The data is collected with respect to metro positions for starting, moving and stopping activity for a single interval *i.e* between two stations. In the Figure 27 shows the sample data collected. The attributes recorded were timestamp, coordinate axis (accX, accY and accZ).



a) *b)*

Figure 27 Application image Start referred in a) and stop referred in b)

In the below Figure 28 shows the sample data recorded from the application for the attributes. After recording the data labelling of the log file is done manually. The information on the storing the data and timestamp is given in Appendix C. In this project a total of 30 data samples collected in the Porto metro yellow line. The data is collected by the single person for various activities (mStart, mMove, mStop) of the metro

1	time	accX	accY	accZ
2	3.32E+14	-1.47962	1.58975	9.619903
3	3.32E+14	-1.52271	1.599327	9.433155
4	3.32E+14	-1.54187	1.575385	9.308656
5	3.32E+14	-1.51792	1.637634	9.246407
6	3.32E+14	-1.59454	1.651999	9.193734
7	3.32E+14	-1.56581	1.608904	9.136273
8	3.32E+14	-1.54187	1.58975	9.203311
9	3.32E+14	-1.53708	1.56102	9.275137
10	3.32E+14	-1.59933	1.565808	9.442732

Figure 28 Sample Data

5.1.2 Data storing code

The data was recorded by accessing the accelerometer coordinates (x,y,z) and timestamp. Below are the few lines of code presented. In the line 3 shows the start of sensor service and in the line 4 code for accessing the accelerometer from sensor service is shown. When broadcast intent is created it include an action string and intent is created for single object. To send extra data with intent, intent's putExtra() method is used in line 9, 10 ,11, 12.

Code:

```
1. Public ServiceHandler(Looper looper ){
2.         super(looper)
3.         mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
4.        mAccelerometer =
mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
5.
6.    }

7. private void broadcastAccelerometerSensor(SensorEvent sensorEvent) {
8.            Intent broadcastIntent = new Intent (ACTION_NEW_ACC_SAMPLE);
9.            broadcastIntent.putExtra(ACC_X_MESSAGE, sensorEvent.values[0]);
10.            broadcastIntent.putExtra(ACC_Y_MESSAGE, sensorEvent.values[1]);
11.           broadcastIntent.putExtra(ACC_Z_MESSAGE, sensorEvent.values[2]);
12.            broadcastIntent.putExtra(TIMESTAMP, sensorEvent.timestamp);
13.            getApplicationContext().sendBroadcast(broadcastIntent);
14.        }
```

In Android program the data is collected from accelerometer sensor which takes gravity along one of the axes. In the program the gyroscope sensor values are also recorded for further analysis to improve the performance.

5.1.3 Data Processing and Analysis

As mentioned in Section 5.1 in data collection approach, the data is collected for aspects of features like mStart, mMove and mStop. An overview of data prior processing is additionally provided in Figure 29. The below figures give the plot of sample data collected.

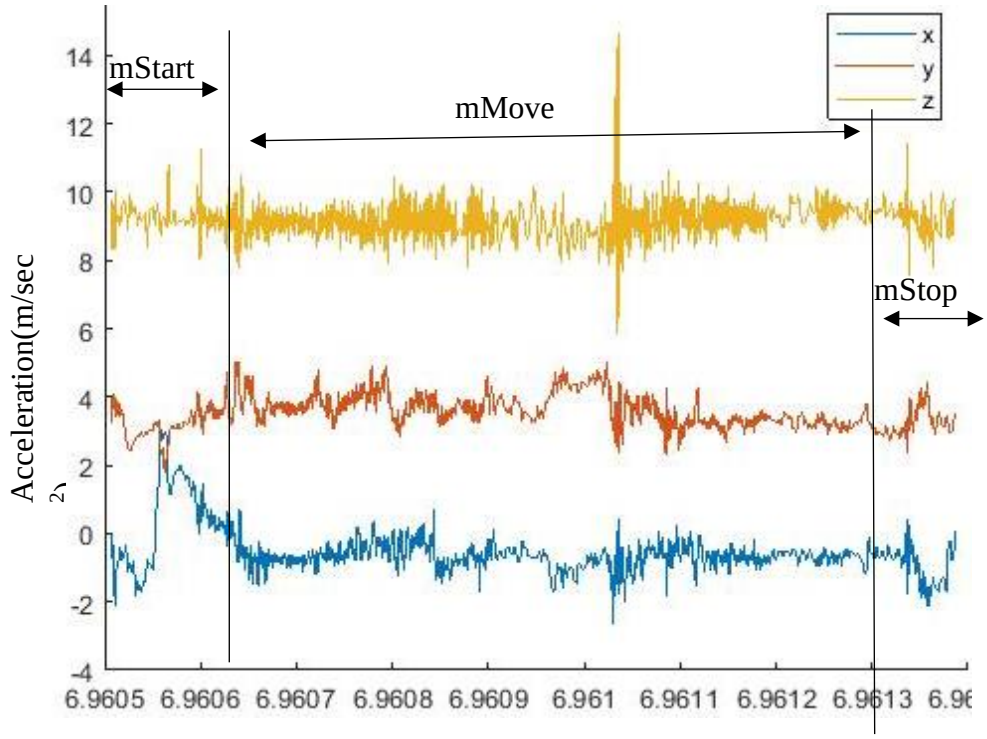


Figure 29 Sample data plot 1

The above plot Figure 29 shows the metro accelerometer raw data collected in x,y and z axis for an interval. The data is collected on the underground metro yellow line from Combatentes to Marques when the metro is moving in forward direction with z-axis pointing towards gravity. Here, in z-axis it has gravity, so it is hanging around 9.8m/sec^2. The x-axis is in the direction of metro with recording the accelerations. From the plot the the transition from start to stop is recognised with the change in acceleration peak. The curve gets raised from start to move and from move to stop the curve gets lower.

5.2 Pre-Processing

Pre-processing refers to transformations applied to the data before feeding it to the algorithm. In data pre-processing the raw data is converted into clean data by applying filters which is feasible for analysis. Some specified Machine Learning model needs information in a specified format, for example Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from data set. Following are the some of the methods used here.

5.2.1 Feature Extraction

A huge amount of raw data is gathered from smartphone sensors. Hence, directly analysing such data would require a lot of time or memory space. A popular approach is to deal is to extract certain important features. Some of the proposed models are time-domain features,

frequency-domain features, wavelets, etc. Selecting suitable features would lead to the increment of the prediction accuracy. In the data collection process, we obtain timestamp. So, using timestamp in this project analysis is done with respect to time-domain features. In time series data features like jitter, kurtosis is extracted in this project. After the extraction the features are copied onto a log file and then it is carried on to pre-processing steps for further classification.

Jitter: Jitter is the undesired deviation in short-term time domain variations in data signal timing.

Skew: It is the measure of asymmetry of the distribution of a real valued random variable.

Kurtosis: it is the measure of peak of the distribution of a true- valued random variable.

5.2.2 Sliding Window

Windowing is a selective sampling technique that allows large datasets to be processed by analysing a small sample. The window selection for setting a sample is based on the amount of data samples. The reasons for using the window algorithm primarily involve gain in time, memory and accuracy. The sliding window approach is based on dividing the data segments into window containing equal number of events based on time, sensor event etc [25]. In this project, using sliding window the data is divided into equal windows and the feature extracted from each window is then used as an input for the recognition algorithms that will associate with the metro activities [22].

5.3 Machine Learning

In this project the sensor data is processed onto Machine learning algorithms. The major part is focussed on machine learning analysis of sensor data as a part of vector system, classification algorithms, predictive analysis, training, testing etc. The machine learning tool used in this project is MATLAB, WEKA and PYTHON.

Training: Training refers to a dataset used to extract feature, train to fit a model and finally validate the set to optimize the model. The process of training a Machine Learning model involves providing an algorithm (*i.e, the learning algorithm*) with training data to learn from.

Testing: Testing refers to the predict new data using the model built on training the set.

5.4 Predictive Model

Predictive Model is a mathematical technique which uses statistics for prediction. It aims to work upon the provided information (dataset) to reach an end conclusion after an event has been triggered. These models make use of classifiers to guess the probability of an outcome

for a given set of input data. Here, the input data considered to develop the model is extracted from the features. The features are extracted from sliding window. The inputs considered are acceleration along x, y, z axis and magnitude to these inputs the neural network is applied and finally a model is obtained. This is the predictive model.

After collecting the data, it is plotted to make analysis with corresponding to different direction, axis (X, Y and Z) then the data is filtered to remove the jitters. A plot corresponding to sliding window is shown in the following chapter 6. The experiment development, tool used for machine learning are discussed in the chapter 4. Finally, a predictive model is obtained.

6 DEVELOPMENT OF PROJECT

In this chapter presents the data carried out in the machine learning choosing various neural networks, analyzing the output model and applying the high accuracy classifier to develop the algorithm.

6.1 Data Acquisition

The orientation of the sensors along a smartphone coordinate system is shown in Figure 24. The structure of accelerometer's data consists of four different fields such as timestamp, accX, accY and accZ. The data is written to the stream message by message as shown in Figure 30 Flowchart showing Copytofile function. After having the data structure, the application is developed in Android Studio. The information from the accelerometer is exported through input event-based file mappings. These devices nodes can be opened by the default file systems calls for reading files. The data can be read from the stream as soon as the sensor measures it.

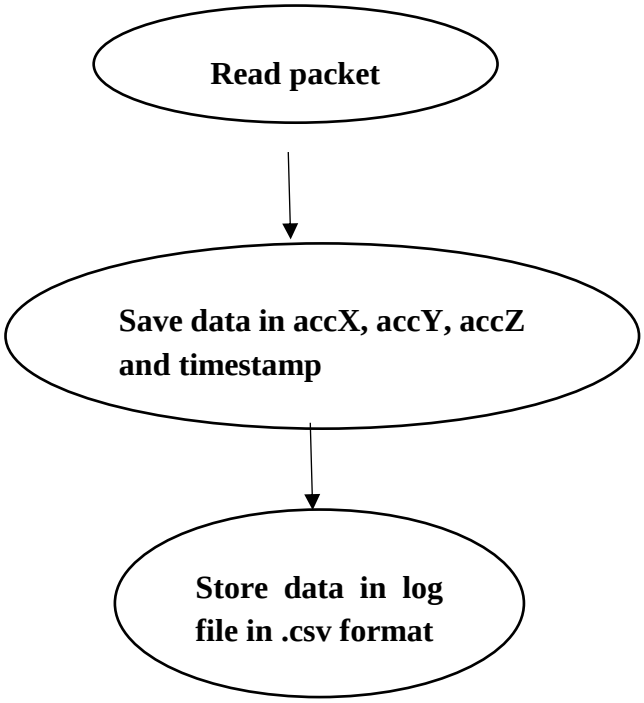


Figure 30 Flowchart showing Copytofile function

In the collecting the data the device nodes are opened. After, the file where the data is going to be saved is opened. Then the specification of attribute (timestamp, accX, accY, accZ) is going to be read is made. The next step is to call copytofile function. The process is explained in a flowchart in Figure 30. It takes the data and copies to the file. In Figure 31 the output data is shown. The data collection process is discussed in chapter 5 section 5.1.

	timestamp	xAxis	yAxis	zAxis
0	6.188280e+14	1.551443	3.083732	9.289502
1	6.188280e+14	1.345541	2.892196	9.074024
2	6.188280e+14	1.403002	2.863465	9.198523
3	6.188280e+14	1.431733	2.849100	9.308656
4	6.188280e+14	1.508347	2.896984	9.327810

Figure 31 Output data

In our scenario, data is collected from accelerometer sensor. The sensor returns four values corresponding to timestamp, x, y and z coordinates. All the data is stored in a log file and then processed onto the pre-processing steps. The raw data used pre-processing steps is a log file collected for an interval between two stations collected for activities mStart, mMove, mStop.

6.2 Raw Data plots

In the project metro is considered as a subject. The accelerations of metro are recorded in x,y and z axis with respect to time. The raw data contains the disturbances, noise of the metro. Below are the raw data plots of metro in different conditions. This helps in analysing the data. When the metro is starting the acceleration curve increases, while moving the acceleration curve gains high and remains constant for short period and then while stopping the curve decreases. The axis lying to the ground has gravity (g = 9.8m/s^2). With reference to change in peak curves the instances for start, move and stop are recognised.

Plot 1: The plot in the Figure 32 shows the data collected on Porto metro line (IPO-Polo Univesitario). The metro travels from ground level to underground. The orientation of the mobile is with z -axis pointing towards gravity y-axis in the direction of metro with acceleration and x-axis with low acceleration. The plot initialises and ends with low acceleration. The moving acceleration increases maintains constant and then decreases. The below is an assumption made to show to analyse the plot. The data contains unequal waves caused due to noise, vibrations. These are filtered further.

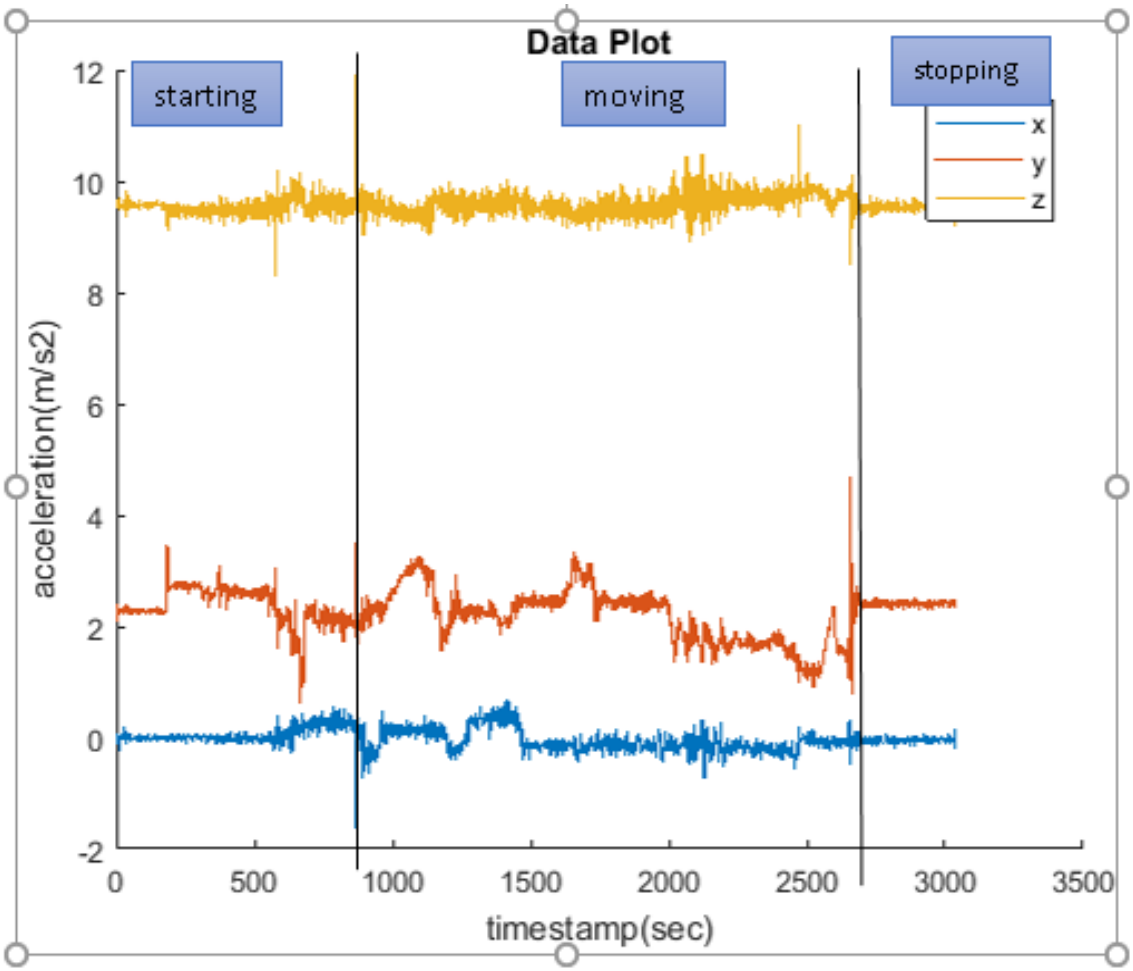


Figure 32 Image plot showing x,y and z axes

Plot 2: The plot in Figure 33 shows the data recorded for underground stations (Trindade-Aliadoss). Here the x- axis is pointing towards gravity, y and z axis has acceleration. While collecting the data the metro initializes with high speed and at certain point braking occurs which caused high disturbances. The orientation of the mobile is with x -axis pointing towards gravity y-axis in the direction of metro with acceleration and x-axis with low acceleration.

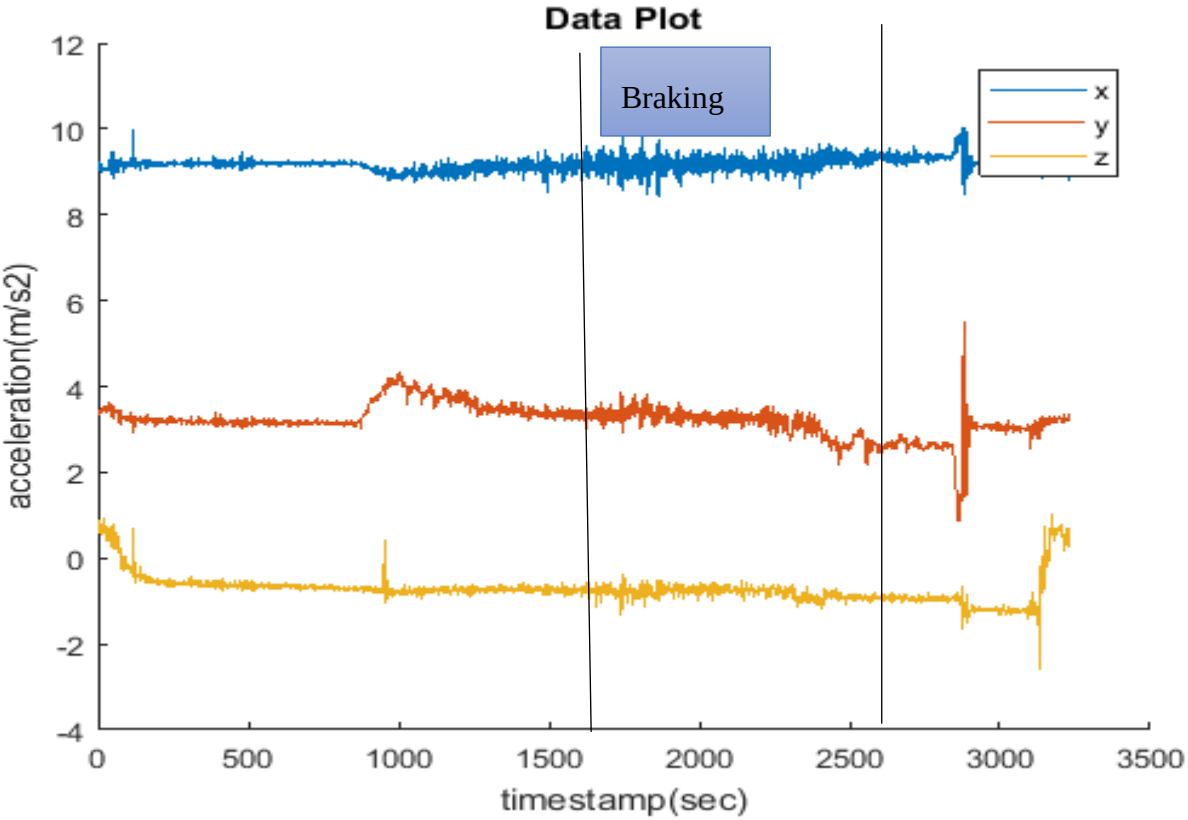


Figure 33 Plot showing individual x,y,z

Plot 3: In the Figure 34shows the data plotted for y-axis facing towards gravity, z axes with acceleration moving in the direction of the metro. The negative acceleration is caused due to uneven surface metro moving downwards (C.Gaia-G.Torres). In the pre-processing steps the vectoral quantities are considered to calculate the magnitude. The orientation of the mobile is with y -axis pointing towards gravity z-axis in the direction of metro with acceleration and x-axis with low acceleration.

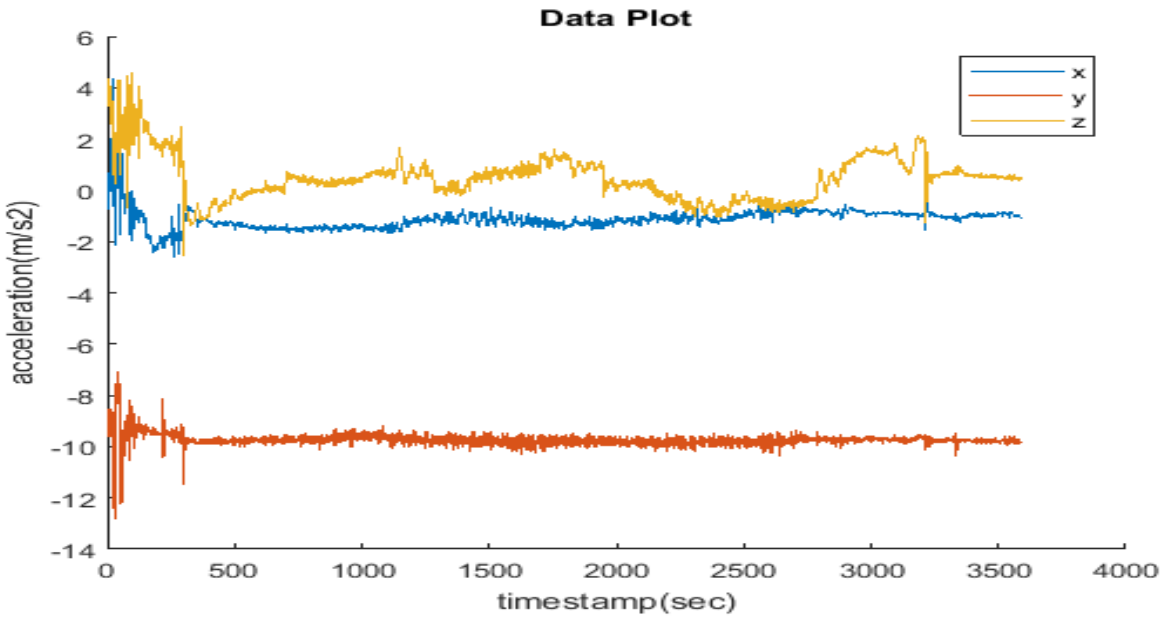


Figure 34 Raw data plot

Plot 4: The below Figure 35 presents the plot in deep to analyse the data along x,y and z axis. The orientation of mobile is considered with ref a) with z-axis pointing towards gravity. The data collected has disturbances, noise, human error. These are filtered and discussed in feature extraction. The x-axis shows the acceleration(m/sec^2) and y-axis shows the timestamp (sec).

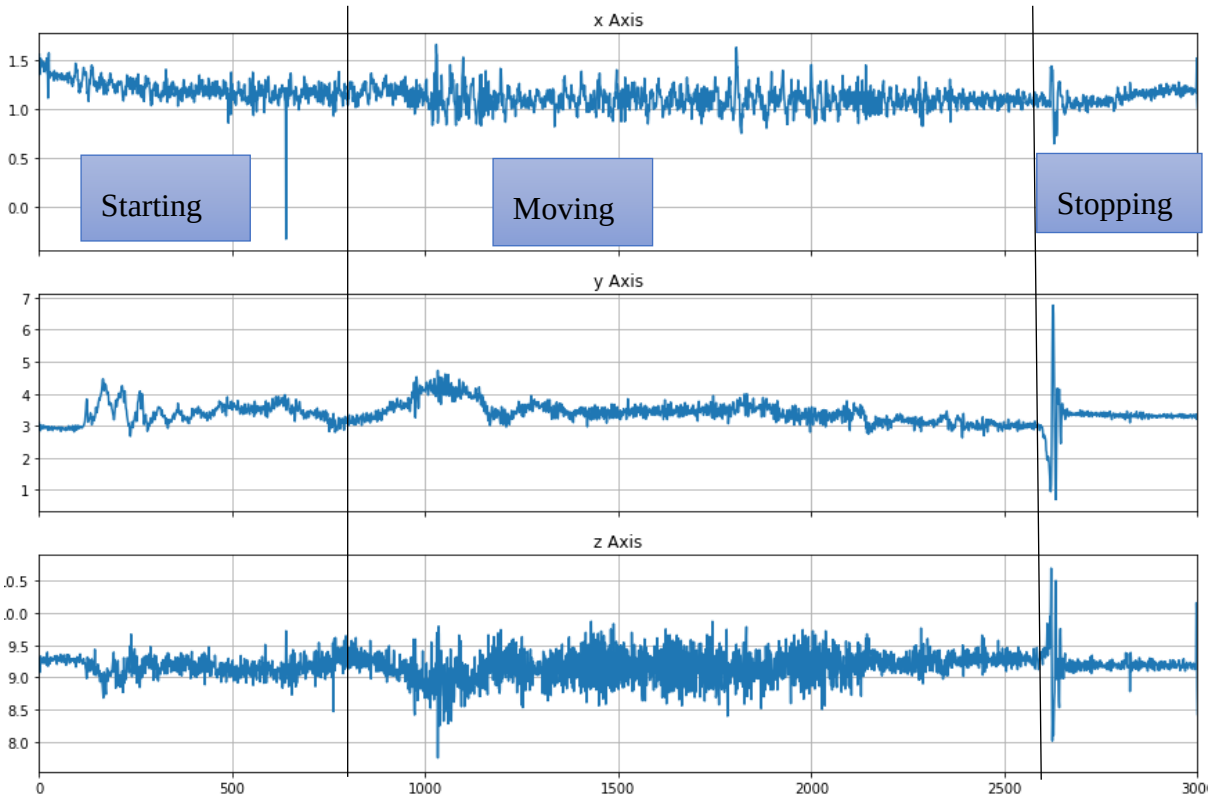


Figure 35 Plot showing individual x,y,z

6.3 Magnitude data

The vectoral calculation is given in the equation 8. In programmatic it is represented in Appendix F using lambda function. Lambda is a small anonymous function can take any number of arguments using one expression. The output of the magnitude vector is shown in Figure 36

syntax: `lambda arguments : expression`

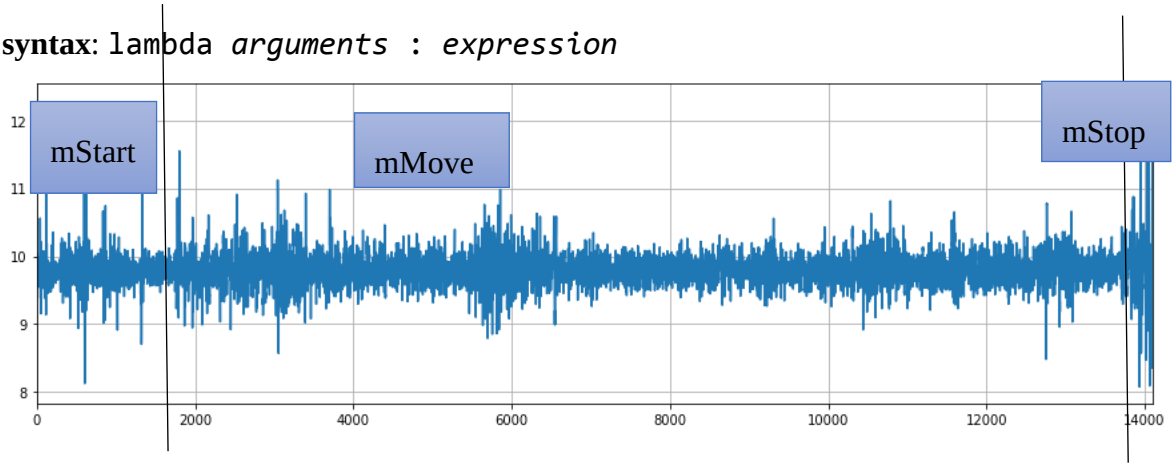


Figure 36 Magnitude Plot

6.4 Sliding Window

The sliding window technique places varying limits on the number of data packets that are sent before pre-processing. The number of data packets is called the window size. The limit on window size vary depending on the rate of data samples recorded. In the Figure 37 the window size is 200 for data samples considered for a total of approx. 3000

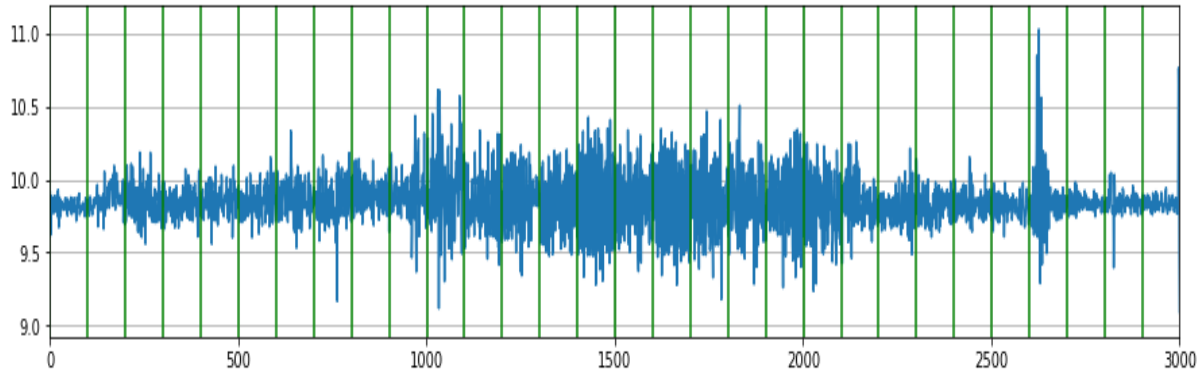


Figure 37 Sliding window plot

6.5 Feature Extraction

Once the sensor values have been pre-processed and transformed into magnitude of accelerations the extraction is done based on peak-based features [26] to identify the end boundary of the peak area. Once the boundaries have been identified statistical set of features are extracted considered in table 3

Table 3 Features considered for classifier

Domain	**Features**
Statistical	Mean, STD, Min, Max
Time	Jitter, kurtosis, Mean-Crossing Rate [27]

6.5.1 Control Chart

A control chart displays measurements of process samples over time. The measurements are plotted together with the specification limits and process-defined control limits. The process can be compared with its specification to see the predicted chart. In the Figure 38 is a control chart used as a monitoring tool which indicates undesirable and systematic characteristics. This control chart is used to discover the variation, so that the process can be adjusted to reduce it. The below plotted control chart shows the magnitude values of the tri-axial accelerometer. Blue circle represents the raw data plotted and the red colour presents the violation of data. This control shows the features extracted by violating the peaks.

After considering the domains the features are extracted from every single window. Here features like

skew: skew is used to compute the skewness of data set. This skewness quantifies the extent to which a distribution differs from normal distribution.

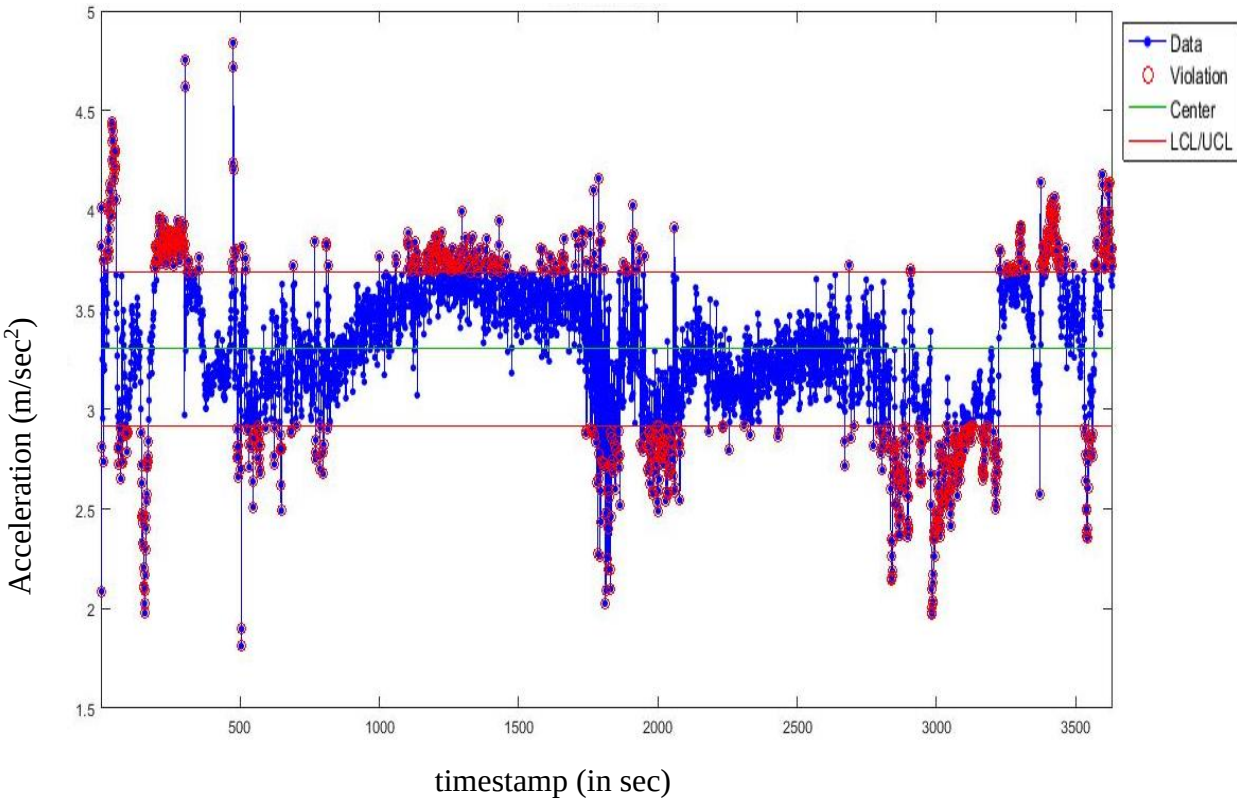


Figure 38 Control Chart

jitter: For a random data when jitter is applied the variables along the categorical axis gets added or subtracted.

kurtosis: kurtosis is a measure of the data are maximum or minimum value relative to normal distribution.

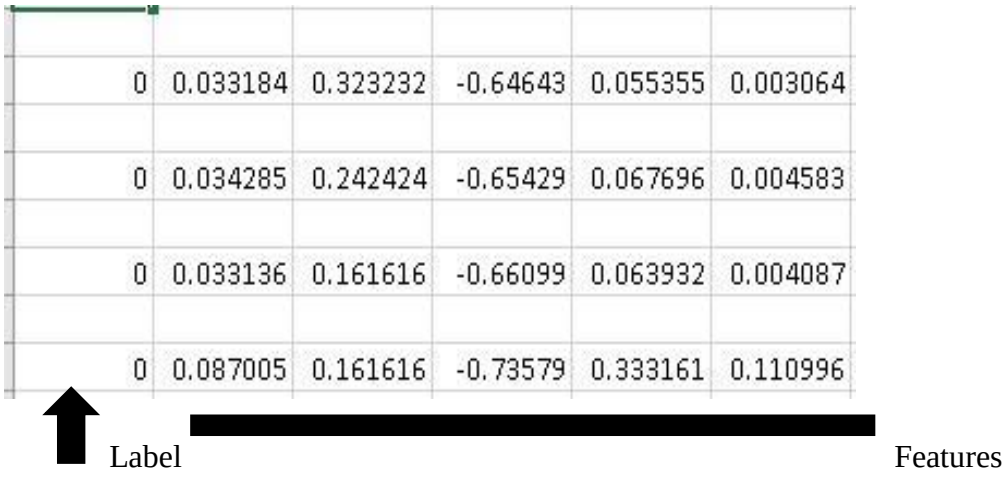


Figure 39 Featured data

In the feature extraction the data samples get adjusted to the normal value. All the samples get remained in the file. The values get added or subtracted to the normal value. Here using jitter, the value gets adjusted, skewness measures the symmetry and kurtosis checks for the peak from -1 to +1. The features considered are used only for extraction of the data. Each

data sample gets remained in the featured dataset. This featured dataset consists of label (mStart, mMove, mStop) and features (x, y, z axis, magnitude). After the extraction of the features from window the data is copied onto a log file shown in Figure 39. Now the data is ready, and the file is imported to apply classification.

6.6 Classification

After the transformation of the data the machine learning algorithm is applied. There is various algorithm considered in this project. The output model is described. Based on the highest accuracy and less training time the best algorithm is chosen. There are various classification algorithms. In this project supervised machine learning is used and the best algorithm is considered from [28] gathered data. The choice of consideration also includes the time of training speed and accuracy. The implemented SVM and K-NN etc., evaluates the features values, takes the appropriate branches until final output is obtained in the classification

6.6.1 Predictive Analytics

Predictive analytics is the procedure of considering huge volumes of data into information. This concept applies complex techniques of classical statistics to credible answers. All predictive analysis involves data, statistical modelling and assumptions. Here supervised machine learning is implemented and carried out as shown in Figure 40

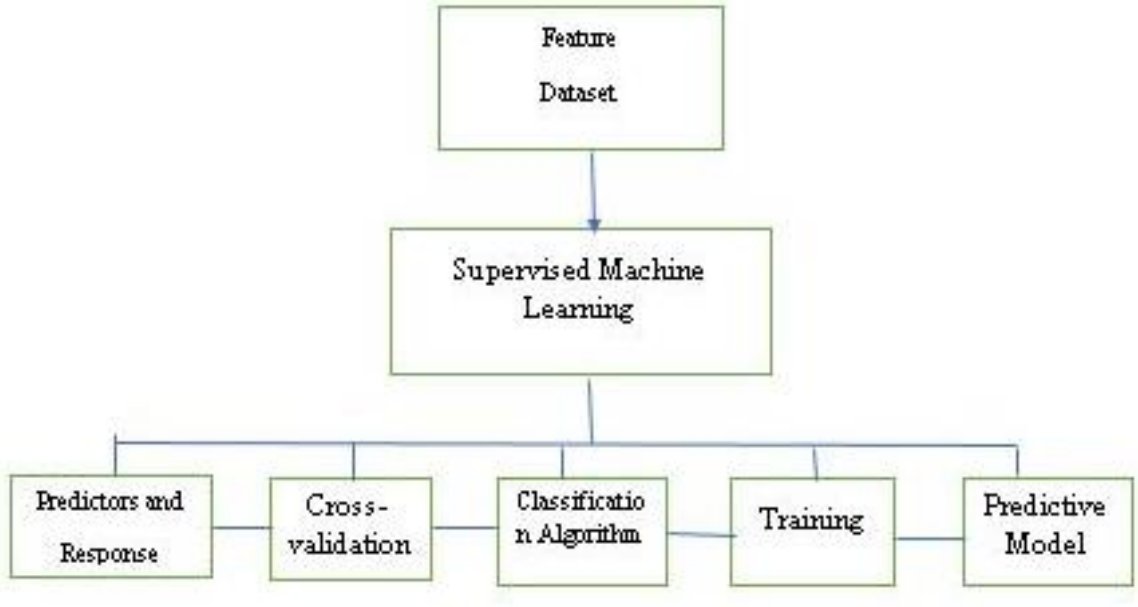


Figure 40 Block diagram data processing using Classification Algorithms

The predictors are the values from the dataset and response is the labelled activity (mStart, mMove, mStop). In all the cases a 10-Fold cross validation is applied which divides the data packets into 10 segments. In training session machine learning algorithm is applied for

training the data and then testing. Finally, a predictive model is obtained. The output analysis for MATLAB, WEKA and PYTHON is discussed below. In all the tools the dataset used is same. It consists of 14100 data samples with segments of mStart (1728), mMove (11336), mStop (1037).

6.6.2 Evaluation in MATLAB

In MATLAB two datasets are prepared of same length (14100 x 5). One dataset is used for training the model. After training the model a code generated to export the model for every classifier. The following are the results obtained for each training classifier

Table 4 MATLAB training results

S.No	Classifier	Accuracy	Training time(sec)
1	**KNN**	**88.5%**	**3.5707**
2	SVM	87.8%	190.44
3	Linear Discriminant	87.2%	3.4805
4	Ensemble Classifier	88.0%	6.4808

After exporting the code, a new dataset of same length is used for testing. It is given as

[trainedModel, validationAccuracy] = trainClassifier (training dataset)

The following are the results obtained for testing dataset (new dataset).

Table 5 MATLAB testing results

S.No	Classifier	Accuracy
1	KNN	82.38%
2	SVM	78.52%
3	Linear Discriminant	76.11%
4	Ensemble Classifier	80.94%

From the training and testing results KNN and Ensemble Classifier (Random Forest) shows the highest accuracy. The reason for highest accuracy is the fraction of predictions the model

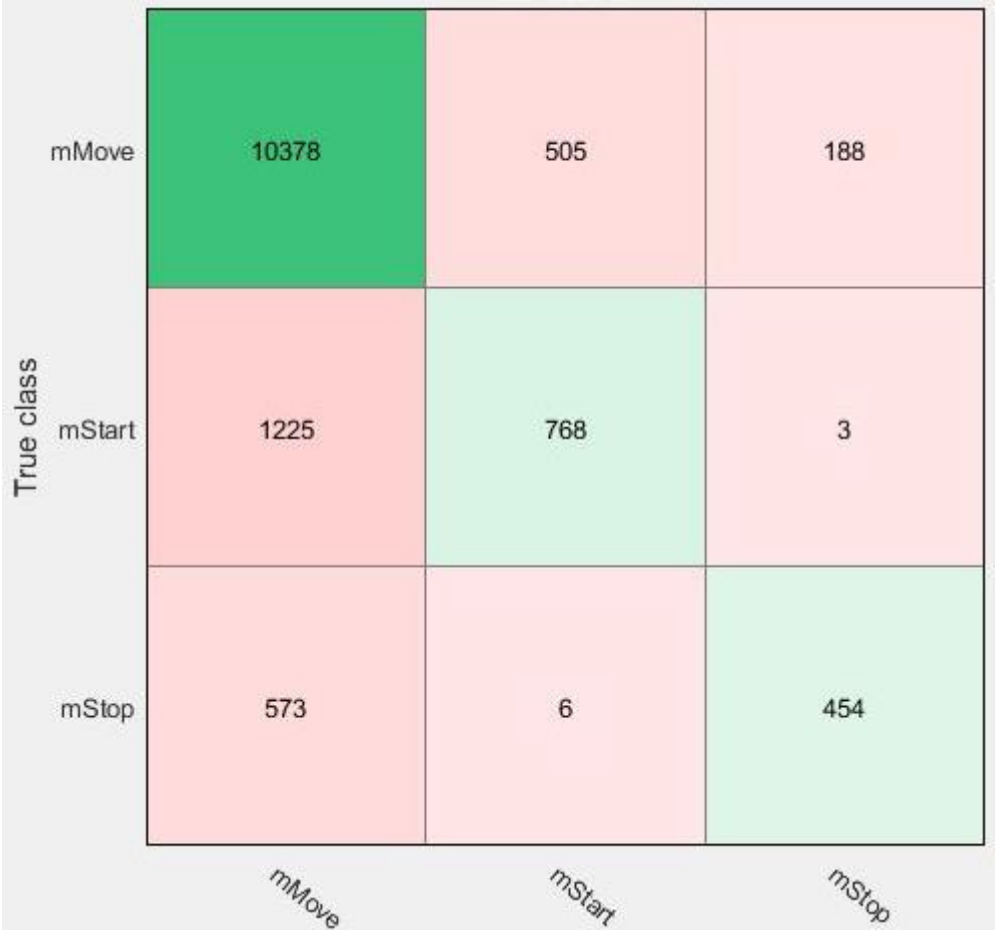
got right from the total number of predictions. It is given in the form of confusion matrix, scatter plot, ROC curve.

6.6.2.1 Confusion Matrix

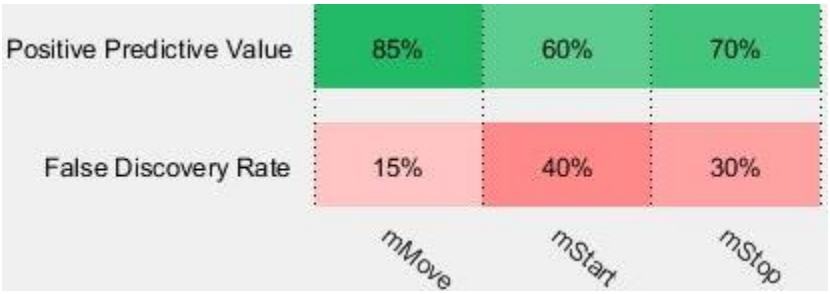
From the confusion matrix the analysis of KNN and RandomForest classifier is explained based on the true classified values.

KNN

After training the dataset of 14100 data samples with segments of mStart (1728), mMove(11336), mStop (1037) with KNN algorithm mStart (454), mMove (10378), mStop (768) colored in green are positively classified, the remaining 367 are confused between mMove and mStart, 125 between mStart and mMove. The green square box shows the true positive classified in Figure 41 a) and Figure 41b) shows the true percentage of the predictive class. Red color shows the false predictive class. In KNN mStart shows the 60%, mMove shows the 85% and mStop shows the 70% truly classified with the overall observations.



a)



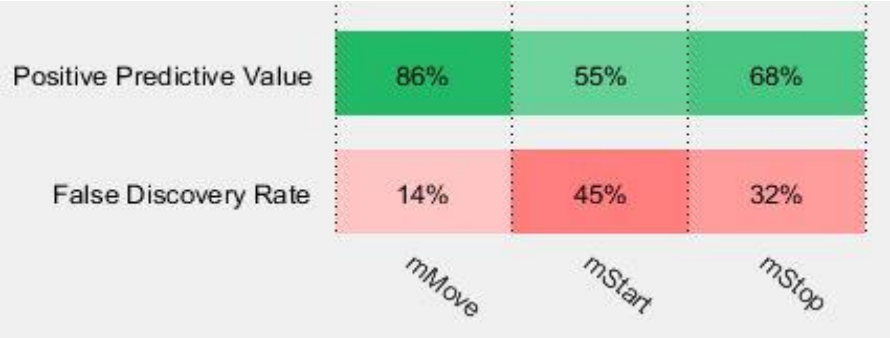
b)

Figure 41 KNN Classifier observations shown in a) percentage of predicted class shown in b)

Random Forest

In the Figure 42 a) shows the confusion matrix for random classifier. Green colour represents the number of true classified and red colour represents the false classified it is represented as higher the accuracy darker the colour. In Figure 42 b) presents the prediction percentage of true value rate and false value rate when Random Forest is applied. The diagonal matrix is the truly classified data samples. In Random Forest mStart shows the 86%, mMove shows the 55% and mStop shows the 68% truly classified with the overall observations. The overall accuracy of the classifier is 88.0%.

a)



b)

Figure 42 Random Forest classifier observations in a) prediction percentage in b)

6.6.2.2 ROC Curves

The ROC curve is a fundamental tool for diagnostic test evaluation. In the ROC curve the true positive rate is plotted in function of the false positive rate for different cut-off points of a parameter. The area under the ROC curve AUC is a measure of how well the parameter can distinguish between two diagnostic group. In the table 5 shows the area under the curve for two classifiers. When compared the results are almost same except mStop. But the true and false rates differ in the curve when compared individually.

Table 6 ROC for KNN and RF

Classifier	**mStart (AUC)**	**mMove (AUC)**	**mStop (AUC)**
KNN	0.87 (0.03, 0.27)	0.91 (0.46, 0.97)	0.94(0.02, 0.64)
Random Forest (RF)	0.87 (0.04, 0.32)	0.91 (0.43, 0.96)	0.94(0.02, 0.58)

In the Figure 43 shows the ROC curve for KNN in the state mMove. The marker shows (0.46, 0.97) 0.46 represents the false positive and 0.97 represents the true positive rates. The AUC is 0.91.

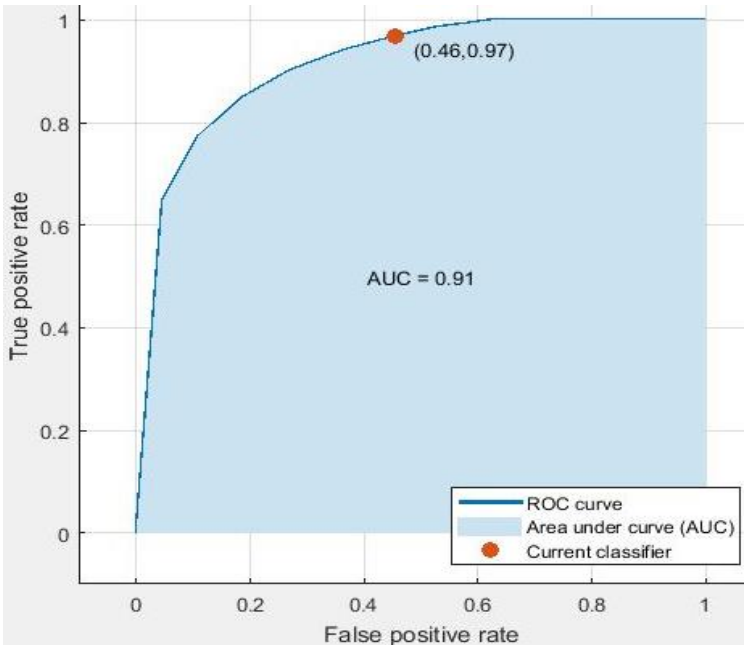


Figure 43 ROC Curve for KNN

In the Figure 44 shows the ROC curve for Random Forest Classifier in the state mMove. The marker shows (0.43, 0.96) 0.43 represents the false positive and 0.96 represents the true positive rates. The AUC is 0.91which is same when compared with KNN but the true positive rate is 1% less than KNN.

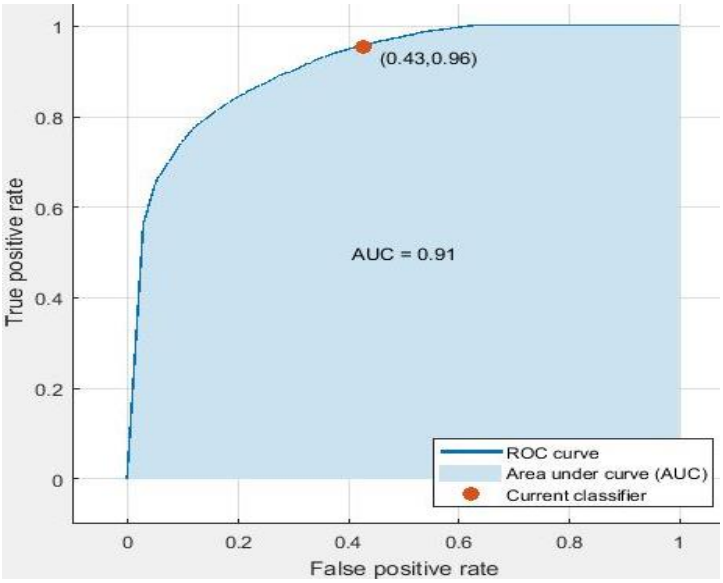


Figure 44 ROC curve in Random Forest

After analyzing the ROC curves in KNN and Random Forest KNN has higher accuracy comparing with Random Forest Classifier in terms of true positive rate. The higher the true positive rate higher the accuracy. The accuracy of mStart and mStop are given in the table 3 a similar response of 1-2% is higher in KNN classifier.

In MATLAB training and testing of the dataset is done. The results obtained with KNN (88.5%) and Random Forest (88.0%) are desirable. But finally, an algorithm in the format of C/C++, Java etc has to be developed. In MATLAB functions like fitctree, fitcknn are not supported to export the code so the analysis is done in Weka and it provides library to develop the program in java, so the experiment is carried out in Weka and its results are discussed below.

6.6.3 Evaluation in WEKA

The evaluation of the each of the classifier was made using Waikato Environment for Knowledge analysis (Weka) software. The classification results in Weka shows a "Detailed Accuracy By Class" and a confusion Matrix. In the table 6 gives the individual accuracy of the classifier.

Table 7 Accuracy of the Classifier

S.No	Classifier Name	Accuracy
1	Random Forest	87.95%
2	Naive Bayes	84.60%
3	**K-NN**	**88.03%**
4	Tree J.48	87.8%
5	SMO	82.98%

The summary of the results performed using various classifiers are shown in table 7 presenting information about the overall classifiers performance.

Table 8 Weighted Average Accuracy By Class

Algorithm	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area
KNN	0.880	0.217	0.879	0.880	0.879	0.673	0.933	0.925
Tree J.48	0.878	0.333	0.863	0.878	0.868	0.590	0.9180	0.912
Random Forest	0.881	0.354	0.862	0.881	0.867	0.590	0.893	0.898
Naïve Bayes	0.769	0.604	0.719	0.769	0.731	0.470	0.790	0.794

Here similar to MATLAB Random Forest and KNN, have showed good accuracy above 86% (with exception of FP rate and MCC). Even though, Random Forest outperformed by KNN and tree J.48 classifier on all performance with 2% higher. The goal here is to find the optimal decision tree by minimizing the prediction error as well as minimizing the number of nodes. There are other classifier like SMO are considered in but the accuracy is low 75% so it is excluded.

In the Figure 45 gives the Detailed Accuracy by Class for Random Forest Classifier. “mStop” is the class with highest TP rate followed by “mMove” and “mStop”. The confusion matrix below is illustrated with 14100 total number of instances. The diagonal has the higher value, i.e., for each activity gives the majority prediction for the correspondent class.

```
=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0,305    0,039    0,524      0,305    0,385      0,339    0,841     0,416     mStart
                 0,958    0,435    0,900      0,958    0,928      0,591    0,891     0,962     mMove
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     mStop
Weighted Avg.    0,881    0,354    0,862      0,881    0,867      0,590    0,893     0,898

=== Confusion Matrix ===

     a     b     c   <-- classified as
   526  1201     0 |     a = mStart
   477 10859     0 |     b = mMove
     0     0  1037 |     c = mStop
```

Figure 45 Detailed Accuracy for Random Forest Classifier

In the Figure 46 shows the Detailed Accuracy by Class for KNN Classifier. Here the TP rate is highes in mMove case followed bt mStop and mStart. The confusion matrix is illustrating

same 14100 number of instances. Here the confusion matrix class is different when compared with Random Forest. The overall accuracy of the classifier is 88.03%.

```
=== Detailed Accuracy By Class ===

                TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                0,638    0,058    0,669      0,638    0,653      0,591    0,921     0,671     mStart
                0,927    0,271    0,917      0,927    0,922      0,665    0,928     0,973     mMove
                0,901    0,007    0,920      0,901    0,910      0,903    0,996     0,962     mStop
Weighted Avg.   0,880    0,217    0,879      0,880    0,879      0,673    0,933     0,925

=== Confusion Matrix ===

    a    b    c   <-- classified as
 1236  702    0 |    a = mStart
  611 8832   80 |    b = mMove
    0  101  920 |    c = mStop
```

Figure 46 Detail Accuracy for KNN Classifier

PairedTTest (corrected): To compare the results of two classifiers with best results a statistical test is done in Weka cool. This paired T-test uses a paired 10-fold cross-validation for both the classifiers, thereby obtaining the accuracy for each classifier when performed with same dataset. The result in figure 41 is the necessary input to proceed with the statistical tests.

The annotation v or * indicates with (v) better result or (*) worse result from the baseline scheme (here random forest) at the significance level specified (0.05). The result of K-NN are statistically worse when compared with the baseline random forest

```
Dataset                   (1) trees.Ran | (2) lazy.
-----------------------------------------------------
dataset_4                 (100)   100.00 |   99.55 *
dataset_5                 (100)    87.96 |   85.66 *
dataset_6                 (100)    81.90 |   78.63 *
-----------------------------------------------------
                                  (v/ /*) |   (0/0/3)
```

Figure 47 Paired T-test. RandomForest identified as (1) and K-NN classifier as (2)

The result of the statistical test shown in Figure 47 gives that Random forest has highest accuracy rate when compared with the KNN performing on different datasets. So, in developing the algorithm in java Random Forest is considered to develop the program.

In MATLAB KNN and Random Forest showed the highest accuracy rate. With the same dataset when the analysis is done in WEKA it showed similar results with KNN (88.03%) and Random Forest with (87.95%). The flexibility with WEKA is it can perform classification on multiple datasets irrespective of length. So, here the confusion is between KNN and Random Forest. After Paired T-Test when performed the test on three different datasets Random Forest good results in all the cases when compared with KNN. Since only one classification algorithm can be implemented to develop the program in java. With reference to paired T-test random forest is chosen to implement in the java program to develop the algorithm.

6.6.4 Evaluation in PYTHON

After the analysis done in MATLAB and WEKA a clear implementation of the process is understood. The reason to use python is easy to code for importing the data with individual files, plotting the data and analyze the plot. In feature selection process implementing the sliding window, calculating the magnitude using lamba function and finally removing the jitters (unwanted signals) became easy by just importing the libraries. With the reference from Weka random forest classification is implemented to develop the model.

After extracting the features, the results are copied to a csv file. To this file classification is applied. Here the cross validation is applied. The goal of cross validation is to test the model's ability to predict new dataset that was not used in estimating. Here in cross validation the data set is split into training and testing dataset. The training set consists of 60% of data and testing set consists of 40% of data. This validation is repeated for 10 folds. Finally, the validation results are combined (mean) to give an estimate of the model's predictive performance.

Here along with RandomForest Dummy Classifier is also implemented. This Dummy Classifier makes predictions using simple rules. This is used as a baseline to compare with random forest success rate.

Code:

```
1. c = RandomForestClassifier()
2. b = DummyClassifier()
3. results = [ ]
4. baselines = [ ]
5. for i in range (0, 10):
6. X_train, X_test, y_train, y_test = train_test_split(X, y,
   test_size=.4)
7. c.fit(X_train, y_train)
8. b.fit(X_train, y_train)
9. res = c.score(X_test, y_test)
10. bas = b.score(X_test, y_test)
```

In the above code RandomForest Classifier (line 1) and Dummy Classifier (line 2) functions are stored into variables b,c. The range i is for iterations. X represents the training set and Y represents testing set. Finally output accuracy for Random Forest is stored in res and for Dummy Classifier in bas. When compiled the program the output is shown below. The program developed in python is shown in Appendix G

Output:

Table 9 Accuracy of the model

Fold	Random Forest Classifier	Dummy Classifier
0	0.8409090909090909	38636363636363635
1	0.75	0.4090909090909091
2	0.8409090909090909	0.5
3	0.7954545454545454	0.4318181818181818
4	0.9090909090909091	0.4772727272727273
5	0.9090909090909091	0.38636363636363635
6	0.8409090909090909	0.4090909090909091
7	0.9772727272727273	0.36363636363636365
8	0.7045454545454546	0.38636363636363635
9	0.8863636363636364	0.5227272727272727
Mean	0.8454545454545455	0.42727272727272725

From the output the overall accuracy of the model is 84.5%. This model predicts the activities of the metro. This algorithm is given in the Appendix F. It has to be further developed to implement the algorithm on a mobile platform

6.7 Predictive Model

Predictive model is a use case driven. Here the model is built using MATLAB, WEKA, PYTHON with a dataset of 14100 samples. The inputs considered are x, y, z axis and magnitude for the response of start, move and stop metro activities. This data set in pre-processing gets filtered after extracting the features. This is procced onto classification process. In MATLAB the highest accuracy of the model is 88.5%. To develop the algorithm in java WEKA is used. Here the predictive model gives 88.03. Here, in WEKA experimenter a classification on random datasets can be passed on to a neural network to predict the overall accuracy of the classifier. From this RandomForest shows the highest accuracy performing on the different datasets. With the flexibility in programming and easy access of libraries in python the predictive model is developed using RandomForest classifier and python has

libraries like TensorFlow to build algorithm for mobile platform. Due to unawareness of TensorFlow only the predictive model is built here using scipy and sklearn. Finally, the predictive model gives an accuracy of 84.5%. To test the predictive model a dataset with single attribute(mMove) is passed the output of the result showed 78% accuracy. The output evaluation model is shown in Appendix G.

7 Conclusion and Future Work

7.1 Conclusions

During this project, the Machine Learning analysis in MATLAB, WEKA and Python are studied. Detection of underground transport maps the user in directing the right destination. Relying on the low power consumption and ubiquity of modern smartphones sensors is the project base explored by the previous researchers. This merely uses accelerometer which is a standard in modern smartphones and a start, move and stop data as input extracted with the metro behaviour. This project opens the analysis of metro activity modes in offline training, testing modes which is put in an algorithm. These algorithms can be further carried to develop the online testing algorithm.

In approach to the project, the working of neural network behaviour is studied and among them K-NN and Random Forest shows the highest accuracy rate with quick training. The Machine Learning tool used are MATLAB, WEKA and Python used in this project. The process of training and testing the data is same but WEKA has freedom to program the algorithm in JAVA which is further integrated into the application. In the evaluation the project achieved the accuracy of 88.5% in MATLAB which is healthy rate when compared with similar projects [4]. In python the program in machine learning became easy after analysing it in MATLAB and WEKA. It has flexible libraries like pandas for plotting time window and extracting features like kurtosis and jitter from time-series data. However, integrating the algorithm into mobile OS would lead to greater impact of the system and achieve prosperous results.

7.2 Future Work

The first step would be the design of online simulator. That is to implement the algorithm on mobile OS. It should be visual, so that everybody could understand it. Collect the data for features like metro turning which helps in avoiding the minimal vibrations addition of gyroscope and pressure sensors which helps in identifying the air pressure in the underground, filtering the disturbances. To include user ease features like directing the user to the right platform, notification of the trip. To get more reliable data. Here it was only one person.

Sensor Fusion – Previous research has shown that other sensors can be used for localization when riding a subway network. The accelerometer [29] [30]and magnetometer [29] have been successfully utilized for location purposes.

Bibliography

[1] "Classification accuracies of physical activities using smartphone motion sensors," *Journal of medical Internet research,* vol. 5, nº e130, p. 14, 2012.

[2] Luca Bedogni, Marco Di Felice, Luciano Bononi, "By Train or By Car?Detecting the User's Motion Type through Smartphone Sensors Data".

[3] Muhammad Shoaib, Hans Scholten, P.J.M. Havinga, "Towards Physical Activity Recognition using smartphone sensors," *10th International Conference on Autonomic & Trusted Computing, Enschede,* 2013.

[4] Thomas Stockx, Brent Hacht, Johannes Schoning, "SubwayPS: Towards Smartphone Positioning in Underground Public Transportation Systems".

[5] Breiman, L.Random Forests, "Machine Learning," p. 45, 2001.

[6] H.Eren, S.Makinist,E.Akin, and A.Yilmaz, "Estimating Driving Behaviour by a Smartphone," 2012.

[7] Thiagarajan, A., Biagioni J., and Eriksson, , "Transit tracking using Smart-phones," *Proceedings of the 8th ACM Conference on Embedded,* pp. 85-89.

[8] Jihoon Yang, Vasant Honavar, "Feature Subset selection Using a Genetic Algorithm," *IEEE Intelligent Systems,* pp. 44-49, 1998.

[9] Hao Xia, Yanyou Qiao, Jun Jian and Yuanfei Chang, "Using Smart Phone Sensors to Detect Transportation Modes," 2014.

[10] Ben-Hur, Asa; Horn, David;Siegelmann, Hava; and Vapnik, Vladimir, "Support Vector Clustering," *Journal of Machine Learning,* vol. 2, pp. 125-137, 2001.

[11] Tavish Srivastava, "Introduction to k-Nearest Neighbours," 2018.

[12] Krishnan, N.C., Cook, D.J, "Activity recognition on streaming sesor data," *Pervasive Mob. Comput,* pp. 138-154, 2014.

[13] “MATLAB Suported Libraries,” [Online]. Available: https://www.mathworks.com/help/simulink/ug/functions-supported-for-code-generation-alphabetical-list.html. [Acedido em 2018].

[14] “Quora,” August 2018. [Online]. Available: https://www.quora.com/Why-is-Python-so-popular-in-machine-learning.

[15] “MATLAB Classication Models,” 2018. [Online]. Available: https://www.mathworks.com/help/stats/train-classification-models-in-classification-learner-app.html#bu3xete. [Acedido em August 2018].

[16] “MATLAB Classifier,” [Online]. Available: https://www.mathworks.com/help/stats/choose-a-classifier.html. [Acedido em August 2018].

[17] EIbe Frank, Mark A.Hall and Ian H.Witten, The WEKA Workbench at 2.1.5, 2016.

[18] S.B. kotsiantis, “Supervised Machine Learning: A Review of Classification Techniques,” 2007.

[19] Eshwari Girish Kulkarni, Raj B. Kulkarni, “WEKA Powerful Tool in Data Mining,” *International Journal of Computer Applications,* 2016.

[20] “matplotlib,” August 2018. [Online]. Available: https://matplotlib.org/.

[21] “statsmodels,” August 2018. [Online]. Available: https://pypi.org/project/statsmodels/.

[22] Nuno Filipe Oliveira Cardoso, “Transport Mode Detection for Elder Care,” 2016.

[23] Ian H. Witten, Eibe Frank, “Data Mining,” *Practical Machine Learning Tools and Techniques.*

[24] “Porto Metro "Line D",” 2018. [Online]. Available: https://en.wikipedia.org/wiki/Porto_Metro.

[25] Narayanan C Krishnan and Diane J Cook, “Activity Recognition on Streaming Sensor Data,” 2015.

[26] Samuli Hemminki, Petteri Nurmi, Sasu Tarkoma, “Accelerometer-Based Transportation Mode Detection on Smartphone”.

[27] Waltenegus Dargie , “Placement Variations and their diagnosis,” 2016.

[28] Pinky Paul., Thomas George, “An Effective Aproach for Human Activity Recognition on Smartphone,” 2015.

[29] Takamasa Higuchi, Hirozumi Yamaguchi, Teruo Higashino, “Tracking Motion Context of Railway Passengers by Fusion of Low-Power Sensors in Mobile Devices”.

[30] Arvind Thiagarajan, James Biagioni, Tomas Gerlich and Jakob Eriksson, “Cooperative transit tracking using smart-phones,” *In Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems,* pp. 85-89, 2010.

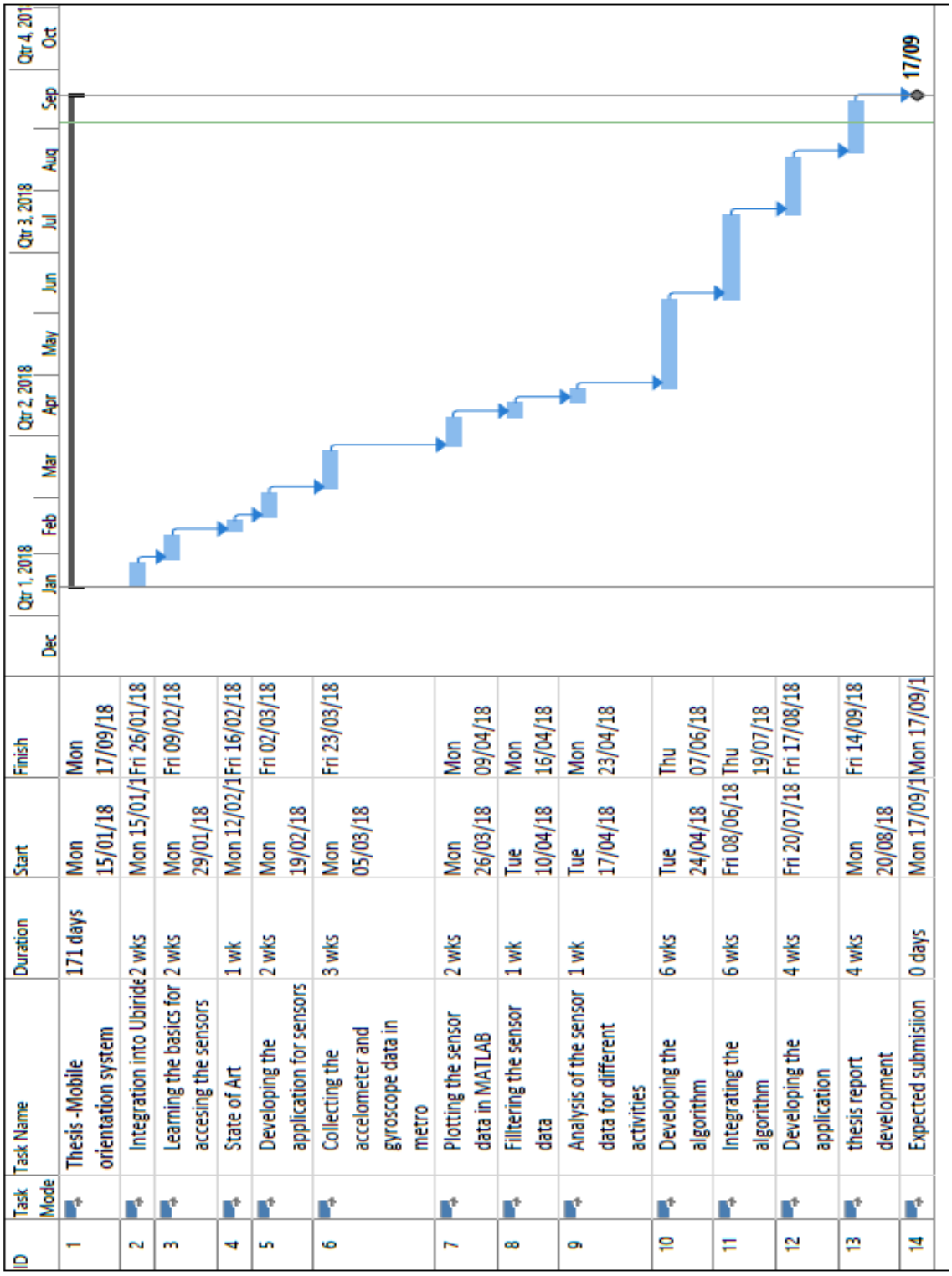
[31] Thomas Stockx, Brent Hecht, Johannes Schoning, “SubwayPS: Towards Smartphone Positioning in Underground Public Transport Systems”.

[32] “Characteristics of Classification Algorithms,” MATLAB, [Online]. Available: https://www.mathworks.com/help/stats/supervised-learning-machine-learning-workflow-and-algorithms.html#bswluh9. [Acedido em August 2018].

[33] “MathWorks Feature-Selection,” MathWorks, August 2018. [Online]. Available: https://www.mathworks.com/help/stats/feature-selection.html.

[34] A Balasubramanian, A LaMarca, “Effectively running continuous monitoring applications on mobile devices using sensor hubs,” 2013.

Appendix A


ID
Task Mode
Task Name
Duration
Start
Finish
1 Thesis -Mobile orientation system 171 days Mon 15/01/18 Mon 17/09/18
2 Integration into Ubiride 2 wks Mon 15/01/1 Fri 26/01/18
3 Learning the basics for accesing the sensors 2 wks Mon 29/01/18 Fri 09/02/18
4 State of Art 1 wk Mon 12/02/1 Fri 16/02/18
5 Developing the application for sensors 2 wks Mon 19/02/18 Fri 02/03/18
6 Collecting the accelometer and gyroscope data in metro 3 wks Mon 05/03/18 Fri 23/03/18
7 Plotting the sensor data in MATLAB 2 wks Mon 26/03/18 Mon 09/04/18
8 Filtering the sensor data 1 wk Tue 10/04/18 Mon 16/04/18
9 Analysis of the sensor data for different activities 1 wk Tue 17/04/18 Mon 23/04/18
10 Developing the algorithm 6 wks Tue 24/04/18 Thu 07/06/18
11 Integrating the algorithm 6 wks Fri 08/06/18 Thu 19/07/18
12 Developing the application 4 wks Fri 20/07/18 Fri 17/08/18
13 thesis report development 4 wks Mon 20/08/18 Fri 14/09/18
14 Expected submisiion 0 days Mon 17/09/1 Mon 17/09/1
Qtr 1, 2018
Qtr 2, 2018
Qtr 3, 2018
Qtr 4, 201
Dec
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
17/09

Appendix B

To gather sensor data for metro, an android application was developed. The application was developed on own with the help of ubirider team to control the log data. Finally, the data is stored in geographical coordinates and collects data for metro start, move and stop. The interface of the application is shown in figure

The logger application logs the following information:

- The timestamp. This is stored in UTC format. The unix time stamp is a way to track time as a running total of seconds. This is very useful to computer systems for tracking and sorted dated information in dynamic and distributed application both online and client side.
- The state of the vehicle is data is collected in a single file. The user has to indicate this manually.

Each time one of these is updated, a new entry is added to the log file.

The logging data is saved to a csv text file. This makes it easy to use the data in Microsoft Excel to do further analysis. To make the file accessible, it is saved on the internal memory of the phone in the *Documents* directory.

Appendix C

Matlab Code

The following is the code generated after training the model. From the results K-NN shows the highest accuracy and the model is generated for K-NN

```
function [trainedClassifier, validationAccuracy] = trainClassifier(trainingData)
% [trainedClassifier, validationAccuracy] = trainClassifier(trainingData)
% returns a trained classifier and its accuracy. This code recreates the
% classification model trained in Classification Learner app. Use the
% generated code to automate training the same model with new data, or to
% learn how to programmatically train models.
%
%  Input:
%      trainingData: a table containing the same predictor and response
%       columns as imported into the app.
%
%  Output:
%      trainedClassifier: a struct containing the trained classifier. The
%       struct contains various fields with information about the trained
%       classifier.
%
%      trainedClassifier.predictFcn: a function to make predictions on new
%       data.
%
%      validationAccuracy: a double containing the accuracy in percent. In
%       the app, the History list displays this overall accuracy score for
%       each model.
%
% Use the code to train the model with new data. To retrain your
% classifier, call the function from the command line with your original
% data or new data as the input argument trainingData.
%
% For example, to retrain a classifier trained with the original data set
% T, enter:
%   [trainedClassifier, validationAccuracy] = trainClassifier(T)
%
% To make predictions with the returned 'trainedClassifier' on new data T2,
% use
%   yfit = trainedClassifier.predictFcn(T2)
%
% T2 must be a table containing at least the same predictor columns as used
% during training. For details, enter:
%   trainedClassifier.HowToPredict


%
```

```
% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'timestamp', 'accX', 'accY', 'accZ'};
predictors = inputTable(:, predictorNames);
response = inputTable.Activity;
isCategoricalPredictor = [false, false, false, false];

% Train a classifier
% This code specifies all the classifier options and trains the classifier.
classificationKNN = fitcknn(...
predictors, ...
response, ...
'Distance', 'Euclidean', ...
'Exponent', [], ...
'NumNeighbors', 10, ...
'DistanceWeight', 'Equal', ...
'Standardize', true, ...
'ClassNames', categorical({'mMove'; 'mStart'; 'mStop'}));

% Create the result struct with predict function
predictorExtractionFcn = @(t) t(:, predictorNames);
knnPredictFcn = @(x) predict(classificationKNN, x);
trainedClassifier.predictFcn = @(x) knnPredictFcn(predictorExtractionFcn(x));

% Add additional fields to the result struct
trainedClassifier.RequiredVariables = {'timestamp', 'accX', 'accY', 'accZ'};
trainedClassifier.ClassificationKNN = classificationKNN;
trainedClassifier.About = 'This struct is a trained model exported from
    Classification Learner R2017a.';
trainedClassifier.HowToPredict = sprintf('To make predictions on a new table, T,
    use: \n  yfit = c.predictFcn(T) \nreplacing "c" with the name of the variable that is
    this struct, e.g. "trainedModel". \n \nThe table, T, must contain the variables
    returned by: \n  c.RequiredVariables \nVariable formats (e.g. matrix/vector,
    datatype) must match the original training data. \nAdditional variables are ignored.
    \n \nFor more information, see <a href="matlab:helpview(fullfile(docroot, "stats",
    "stats.map"), "appclassification_exportmodeltoworkspace")">How to predict using
    an exported model</a>.');

% Extract predictors and response
% This code processes the data into the right shape for training the
% model.
inputTable = trainingData;
predictorNames = {'timestamp', 'accX', 'accY', 'accZ'};
predictors = inputTable(:, predictorNames);
```

```
response = inputTable.Activity;
isCategoricalPredictor = [false, false, false, false];

% Perform cross-validation
partitionedModel = crossval(trainedClassifier.ClassificationKNN, 'KFold', 10);

% Compute validation predictions
[validationPredictions, validationScores] = kfoldPredict(partitionedModel);

% Compute validation accuracy
validationAccuracy = 1 - kfoldLoss(partitionedModel, 'LossFun', 'ClassifError');
```

Description:

The above code is generated after training the algorithm. When looking into broader way in MATLAB after importing the data it is processed onto classification learner application. The process includes

step 1: dataset (length 14100 x 5)

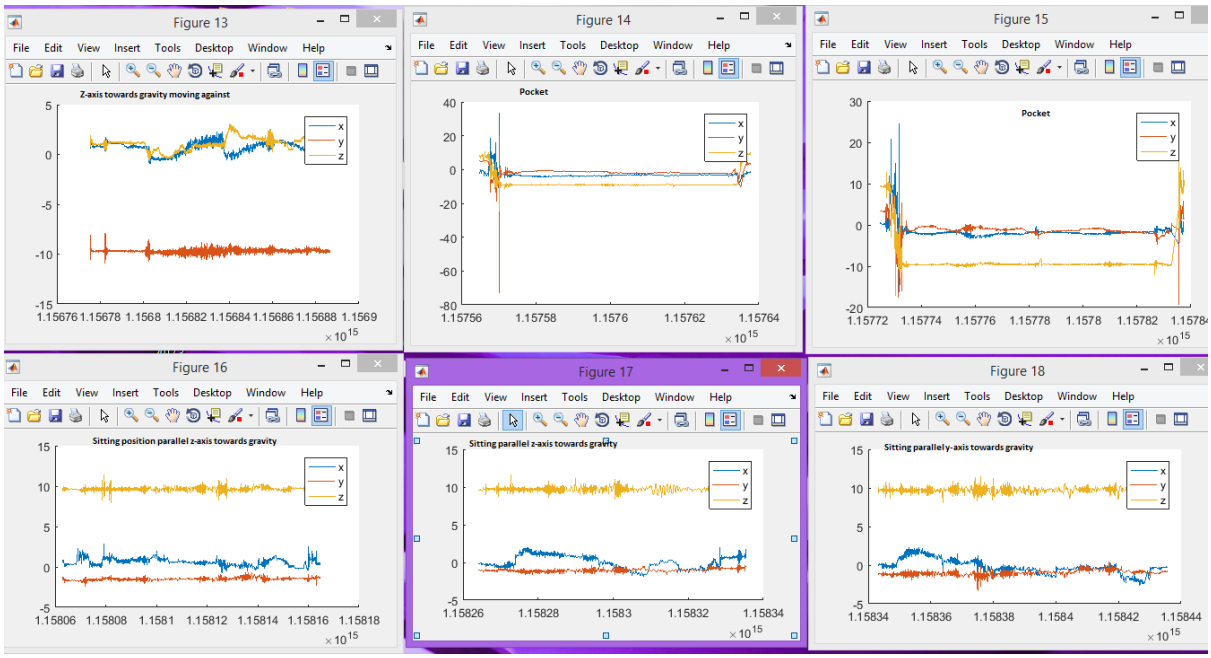
step 2: select predictors (timestamp, accX, accY, accZ) and responses (mStart. mMove, mStop) step

3: Choose a validation method (Cross Validation = 10 folds).

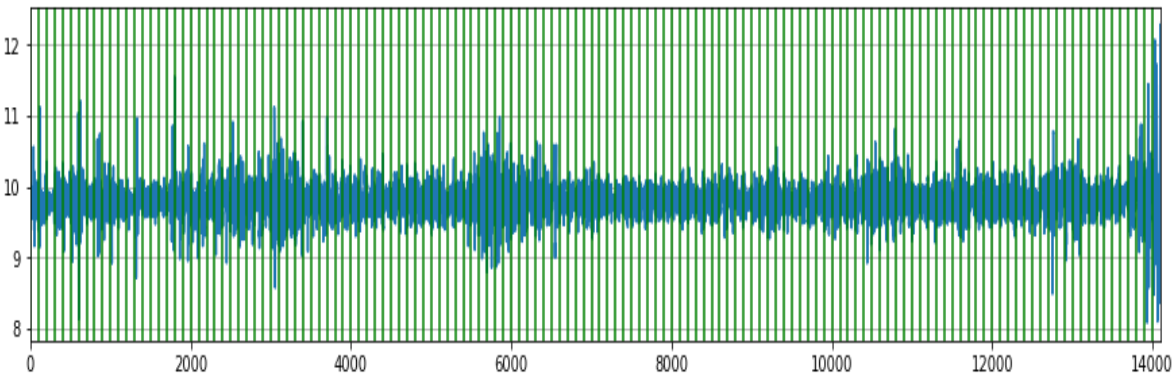
In the classification Learner for the above code KNN is applied for training the data. After training the data MATLAB has feature to generate the code. After training is done a new dataset of equal length is used for testing the dataset. It is mentioned in the line no 67 to make predictions for new data. This is the machine learning model built using MATLAB 2017a. The result obtained when training the data is 88.5% accuracy and for the testing data the accuracy is 82.38%.

Appendix D

Raw data Plots



Sliding Window



Appendix E

Java algorithm developed in WEKA

```
1.  //load datasets
2.  DataSource source = new DataSource("/home/inspiron/Desktop/DataSets/dataset_5.arff");
3.  Instances dataset = source.getDataSet();
4.  //set class index to the last attribute
5.  dataset.setClassIndex(dataset.numAttributes()-1);
6.  //create and build the classifier!
7.  //J48 tre = new J48();
8.  //tre.buildClassifier(dataset);
9.  //IBk neural = new IBk();
10. //neural.buildClassifier(dataset);
11. //SMO svm = new SMO();
12. //svm.buildClassifier(dataset);
13. NaiveBayes naive = new NaiveBayes();
14. naive.buildClassifier(dataset);
15. //RandomForest rf = new RandomForest();
16. //rf.buildClassifier(dataset);
17. //MIPolyKernel mip = new MIPolyKernel();
18. //mip.buildClassifier(dataset);

19. Evaluation eval = new Evaluation(dataset);
20. Random rand = new Random(1);
21. int folds = 10;

22. //Notice we build the classifier with the training dataset
23. //we initialize evaluation with the training dataset and then
24. //evaluate using the test dataset

25. //test dataset for evaluation
26. DataSource source1 = new DataSource("/home/inspiron/Desktop/DataSets/dataset_6.arff");
27. Instances testDataset = source1.getDataSet();
28. //set class index to the last attribute
29. testDataset.setClassIndex(testDataset.numAttributes()-1);
30. //now evaluate model
31. eval.evaluateModel(naive, testDataset);
32. eval.crossValidateModel(naive, testDataset, folds, rand);
33. double Correct = eval.correct();
34. String strSummary = eval.toSummaryString();
35. //System.out.println(eval.toSummaryString("Evaluation results:\n", false));
36. strSummary.split("a");
37. System.out.println(strSummary);
38. System.out.println("a");


39. // Print per class results
40. String resPerClass = eval.toClassDetailsString();
```

```
41. System.out.println(resPerClass);

42. // Get the confusion matrix
43. String cMatrix = eval.toMatrixString();
44. System.out.println(cMatrix);
```

Appendix F

Python algorithm

/* Reading the data file***/**

```
1. COLUMNS = ['timestamp', 'xAxis', 'yAxis', 'zAxis']
2. STARTING = pd.read_csv('../Data/test2.csv', header=None,
   names=COLUMNS)[:5000]
3. MOVING = pd.read_csv('../Data/mMove.csv', header=None,
   names=COLUMNS)[:3000]
4. STOPPING = pd.read_csv('../Data/mStop.csv', header=None,
   names=COLUMNS)[:3000]
5. POCKET = pd.read_csv('../Data/pocket_1462487242397.csv', header=None,
   names=COLUMNS)[:3000]
6. ON_TRAIN = pd.read_csv('../Data/mMetro_1462518004872.csv', header=None,
   names=COLUMNS)[:3000]


7. STARTING.head()
```

/*Making the plot*/

```
1. def plot_axis(ax, x, y, title):
2. ax.plot(x, y)
3. ax.set_title(title)
4. ax.xaxis.set_visible(False)
5. ax.set_ylim([min(y) - np.std(y), max(y) + np.std(y)])
6. ax.set_xlim([min(x), max(x)])
7. ax.grid(True)


8. def plot_activity(activity):
9. fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(15, 10), sharex=True)
10. plot_axis(ax0, activity['timestamp'], activity['xAxis'], 'x Axis')

plot_axis(ax1, activity['timestamp'], activity['yAxis'], 'y Axis')

plot_axis(ax2, activity['timestamp'], activity['zAxis'], 'z Axis')
```

```
plt.subplots_adjust(hspace=0.2)

plt.show()
```

/* Magnitude Calculation*/

Magnitude Calculation: Here in the lambda function is given with an argument. The parameters are passed, and lambda calculates the square of given integer value.

```
1. Import math
2. def magnitude(activity):
3. accX = activity['xAxis'] * activity['xAxis']
4. accY = activity['yAxis'] * activity['yAxis']
5. accZ = activity['zAxis'] * activity['zAxis']
6. a = accX + accY + accZ
7. m = a.apply(lambda x: math.sqrt(x))
8. return m
```

/* defining the window size*/

```
1. def windows(df, size=200):
2. start = 0
3. while start < df.count():
4. yield start, start + size
5. start += (size / 2)
```

/*Extracting the features*/

```
1. def jitter(axis, start, end):
2. j = float(0)
3. for i in xrange(start, min(end, axis.count())):
4. if start != 0:
        a. j += abs(axis[i] - axis[i-1])
5. return j / (end-start)
```

```
6. def mean_crossing_rate(axis, start, end):
7. cr = 0
8. m = axis.mean()
9. for i in xrange(start, min(end, axis.count())):
10. if start != 0:
        a. p = axis[i-1] > m
        b. c = axis[i] > m
        c. if p != c:
```

```
        d.  cr += 1
11. return float(cr) / (end-start-1)
```

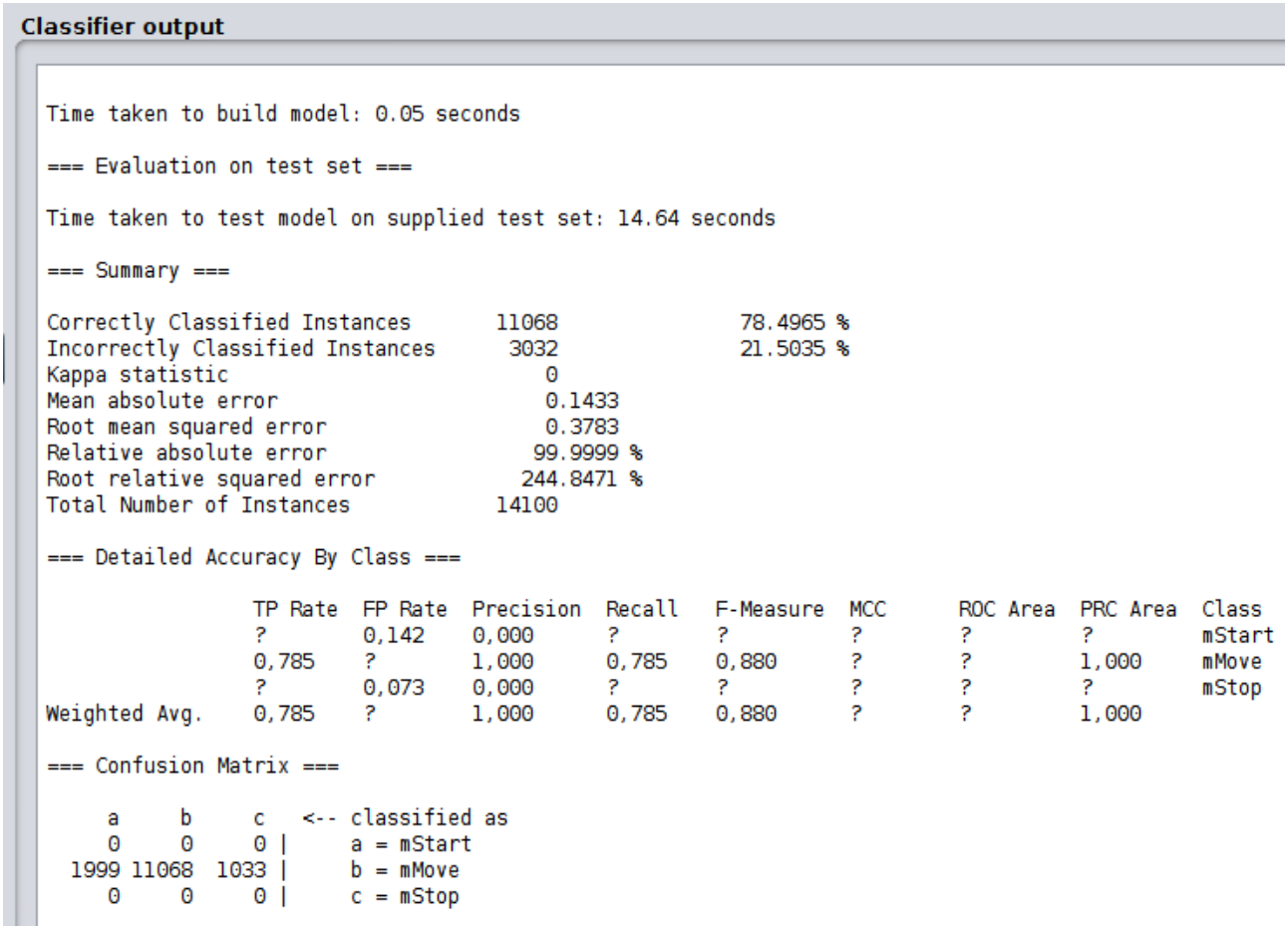
/*Applying the classifier*/

```
1. c = RandomForestClassifier()
2. b = DummyClassifier()
3. results = [ ]
4. baselines = [ ]


5. for i in range (0, 10):
6. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4)
7. c.fit(X_train, y_train)
8. b.fit(X_train, y_train)
9. res = c.score(X_test, y_test)
10. bas = b.score(X_test, y_test)
11. print 'Fold', i, res, bas
12. results.append(res)
13. baselines.append(bas)
```

Appendix G

Explanation: In WEKA application after the training is done I have created a duplicate file of equal length with only one attribute(mMOVE) the accuracy and confusion matrix are shown in the figure



Classifier output

```
Time taken to build model: 0.05 seconds

=== Evaluation on test set ===

Time taken to test model on supplied test set: 14.64 seconds

=== Summary ===

Correctly Classified Instances       11068               78.4965 %
Incorrectly Classified Instances      3032               21.5035 %
Kappa statistic                          0
Mean absolute error                      0.1433
Root mean squared error                  0.3783
Relative absolute error                 99.9999 %
Root relative squared error            244.8471 %
Total Number of Instances            14100

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 ?        0,142    0,000      ?        ?          ?        ?         ?         mStart
                 0,785    ?        1,000      0,785    0,880      ?        ?         1,000     mMove
                 ?        0,073    0,000      ?        ?          ?        ?         ?         mStop
Weighted Avg.    0,785    ?        1,000      0,785    0,880      ?        ?         1,000

=== Confusion Matrix ===

     a     b     c   <-- classified as
     0     0     0 |     a = mStart
  1999 11068  1033 |     b = mMove
     0     0     0 |     c = mStop
```