TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Mahmoud Hany Mahmoud Elbayoumy 165518IVEM

MOBILITAPP PROJECT: USER TRANSPORTATION ACTIVITY RECOGNITION VIA MOBILE DEVICE SENSORS

Master's Thesis

Supervisors: Alar Kuusik

Dr., Senior researcher of T.J. Seebeck Dept of Electronics (TUT)

Mónica Aguilar Igartua

Associate Professor in the Department of Networking Engineering (UPC)

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Mahmoud Hany Mahmoud Elbayoumy 165518IVEM

MOBILITAPP PROJEKT: LIIKLEJA TRANSPORDIKASUTUSE TUVASTAMINE MOBIILSE SEADME SENSORITE ABIL

Magistritöö

Juhendaja: Alar Kuusik

tehnikateaduste doktor infotehnoloogia alal, vanemteadur (TTÜ)

Mónica Aguilar Igartua

Võrgutehnoloogiate osakonna dotsent (UPC)

Tallinn 2018

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Mahmoud Elbayoumy

14.05.2018

Abstract

Current thesis presents my developments of algorithms and software for detecting means of transportation of citizens using motion data from the sensors built in Android smart phones.

First, I worked on adding sensors data logging functionalities to an Android application called MobilitApp. Second, I worked on logging sensors data via using MobilitApp's new functionality. Third, using the data collected, I worked with machine learning concepts to be able to detect the specific transportation activities. Eventually, I assessed multiple solutions to integrate the machine learning logic implementations with Android operating system.

This thesis is written in English and is 41 pages long, including 5 chapters, 17 figures and 8 tables.

Annotatsioon
Mobilitapp projekt: liikleja transpordikasutuse tuvastamine mobiilse seadme sensorite abil

Käesolev lõputöö kirjeldab minu arendustöid algoritmide ja tarkvara vallas, mille eesmärgiks on kodanike transpordivahendite kasutuse tuvastamine Android nutitelefonides sisalduvate liikumisandurite abil.

Esmalt lisasin Android tarkvararakendusele MobilitApp sensorandmete logimise funktsionaalsuse. Teiseks tegelesin sensorandmete kogumisega kasutades seda MobilitApp uut funktsionaalsust. Kolmandaks, kasutades kogutud andmeid, tegelesin masinõppe meetoditega tuvastamaks spetsiifilisi transpordiaktiivsusi. Lõpuks uurisin erinevaid lahendusi integreerimaks masinõppealgoritmide implementatsioone Android operatsioonisüsteemiga Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 41 leheküljel, 5 peatükki, 17 joonist, 8 tabelit.

List of abbreviations and terms

API	*Application programming interface*
APK	*Android Package. It is the file generated after developing an Android phone application, used in its distribution and installation.*
App	*Software Application*
ATM	*Autoritat del Transport Metropolità*
m/s^2	*Meters per second squared*
min	*Minutes*
ms	*Milliseconds*
OS	*Operating System*
PC	*Personal Computer*
PCA	*Principal Component Analysis*
s	*Seconds*
SCs	*Smart Cities*
SDK	*Software Development Kit. It is a set of development tools assists in building certain software.*
SVM	*Support Vector Machine*
TUT	*Tallinn University of Technology*
UI	*User Interface*
UPC	*Universitat Politècnica de Catalunya*

Table of contents

List of figures

List of tables

1 Introduction

Smart cities (SCs) is one of the leading sensing and connectivity technology deployment concepts nowadays. Everything around us is turning to be a smart object. One of the major tracks of smart cities solutions is transportation by its all means. Detecting citizens transportation activities (driving, cycling, walking, running or in a certain public transportation) will assist in improving the transportation map of any city.

MobilitApp [1] is a project hosted in Universitat Politècnica de Catalunya (UPC) [2], Barcelona, which focuses on the smart thing that is always accompanying citizens during their daily transportation journeys, which is the smart phone. This project mainly focuses on adding new functionalities to an existing Android software application (app) called MobilitApp. New added functionalities are: collecting data from the mobile phone's sensors to be passed afterwards to a machine learning algorithm to be able to detect the means of transportation the phone holder is using. In our case, MobilitatApp developers consider Barcelona in Spain as our prototype case study.

Our ultimate goal is to offer MobilitApp to companies responsible for public transportation, such as Autoritat del Transport Metropolità (ATM) in Barcelona, automated bike rental companies, etc. For such service providers, information such as what the citizens use to transport from certain place to another, what are the routes that citizens always use and what are the hot destinations of the city, such information will be of a great deal in enhancing the city transportation map.

In this part of the project, the author's work was mainly focused on enhancing MobilitApp application to be able to collect data from mobile phone sensors, and upload it to a remote server to be stored and analysed. Afterwards a machine learning algorithm was set to detect the activity done by the user based on the data collected. It worth mentioning that Google has created and Android Activity Recognition API [3], which allows to classify certain user activities like walking and driving a car, but it just detects a few motion activities. Present thesis describes author's work to classify different modes of transport and therefore extends the present state-of-art of Android Activity Recognition API. I managed to capture the output for Android Activity Recognition API for the same activities during progressing in the project and I will try to show how the predictions of Android versus mine versus the actual ones were.

1.1 Structure of Work

This work is structured in 5 main chapters. Chapter 1 is the current introduction. Chapter 2 focuses on the market overview and current solutions already in business and the scope of each of them. Chapter 3 where I focus on my objective, obstacles and my plan to overcome them. Chapters 4 illustrates the procedures followed to reach my final target. Chapter 5 contains the results, the discussion around them and the future work.

2 Transportation Applications Market Overview

Over the years and since smartphones become in the hands of everybody, mobile applications started to invade the market, and their existence started to take over major part of the web applications' role. Currently, the competition in the field of smart transportation and routes applications is very fierce. Each and every company in the market is trying to offer the software application (app) with the best features to gain the biggest market share and to be a role model in the industry. This section shows some examples of those apps to show the current direction of the business and sense the market flavour. As mentioned before there are a lot of apps in the market, but those apps presented here are meant to illustrate the different concepts which a smart traffic app nowadays can work with.

2.1 Market Solutions

There are several smartphone applications dedicated for smart traffic solutions in order to enhance the citizens' experience on the road and facilitate their daily life. In the following, a selection of those application focusing on their diversified scopes.

2.1.1 Moovit [4]

Platforms: Web, Android, iOS

Service locations: over 2000 cities

Moovit is adopting the same concept of Google maps [5] where it shows the user the options of routes to his/her destination, the estimated trip time for each route, the best route of them, the means of transportations can be used (including public transportation stations, route numbers and timetables).

Moovit also offers reselling solutions through their widgets and white-label solutions for companies that want to implement an interactive maps in their applications.

2.1.2 My Smart Route [6]

Platforms: Web, Android

Serving locations: Worldwide

My Smart Route is meant to be for users who work in the delivery business. It enables the user to plan and model the daily delivery routes. The user enters details of his/her customers like addresses, time windows, departure time, then the system applies a route optimization algorithm and create optimized, feasible and cost-effective multi-drop routes.

2.1.3 Waze [7]

Platforms: Android, iOS

Service locations: Worldwide

Waze is meant to be for private vehicle owners. Users open the application and enter the destination address then start driving while the app is opened. The app on its own starts to collect traffic and other road data. Also, users can share road reports on accidents, police traps, or any other hazards along the way which help others users during their trips.

2.1.4 By2ollak [8]

Platform: Android, iOS

Serving locations: Cairo, Egypt

By2ollak is meant to be for private vehicle owners. It works with the concept of reporting where each driver can report the status of the road he/she is driving in and those reports are shared with other users. It is somehow like Waze, but without any smart algorithm for collecting the traffic data.

2.1.5 Android Activity Recognition APIs [3]

Platform: API for Android apps

Serving locations: Worldwide

Google Activity recognition is an API developed by Google to be used in Android application. This API uses the built in Android phone sensor to differentiate between 8 user activities, shown in Table 1. The API output is the probabilities for each activity to be done by the user in a particular period of time.

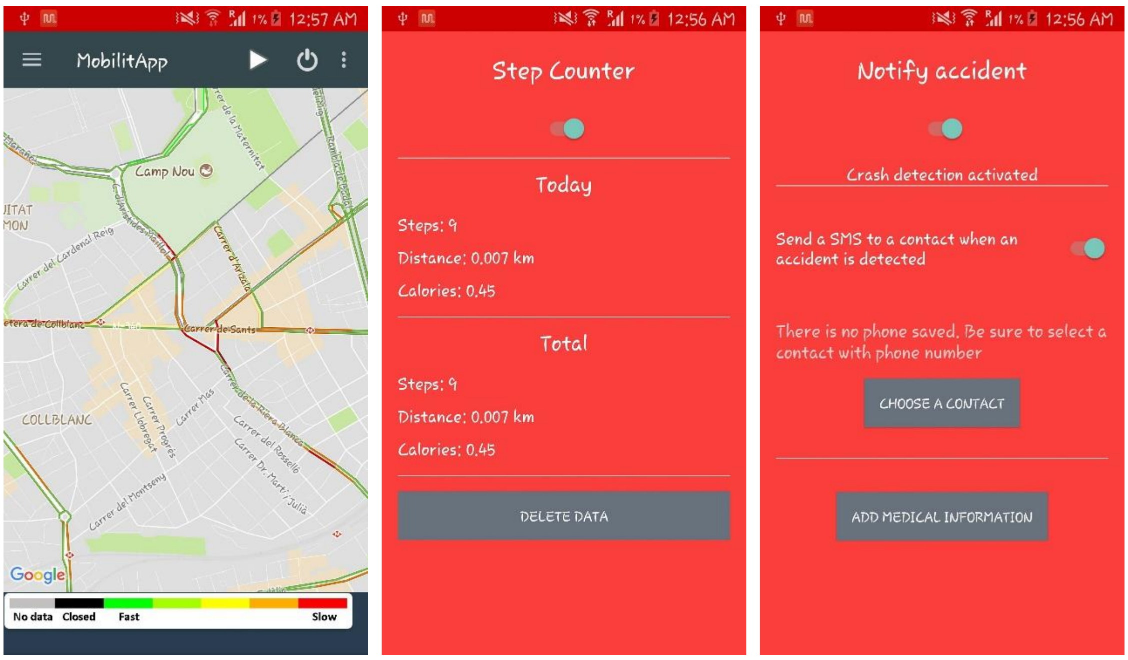
Table 1 Activity Recognition API Detected Activities

Activity Name	**Activity Description**
IN_VEHICLE	The device is in a vehicle, such as a car.
ON_BICYCLE	The device is on a bicycle.
ON_FOOT	The device is on a user who is walking or running.
RUNNING	The device is on a user who is running.
STILL	The device is still (not moving).
TILTING	The device angle relative to gravity changed significantly.
UNKNOWN	Unable to detect the current activity.
WALKING	The device is on a user who is walking.

2.2 MobilitApp [9]

MobilitApp project started in 2014 in Universitat Politècnica de Catalunya (UPC) [2], Barcelona and still ongoing till now. MobilitApp main target is to enhance the citizens' daily transportation experience and to improve the public transportation over the city of Barcelona. It is the collaboration of many students' work. The major difference I am targeting to add to MobilitApp over other apps in the market is the focus on activity recognition of the users. I want to detect how the user is transporting during his day without any intervention of the user himself, so I could reach a fully automated app having an Artificial Intelligence concept. Android Activity Recognition API offers the same utility, however it cannot distinguish between different means of transportations (car, metro, train, bus, etc.). Other services like route advising and traffic density can be served as an auxiliary services. However, our main scope here is to reach a fully automated platform to inform the user without the user informing it or inserting any input.

MobilitApp currently offers multiple useful services as shown in Figure 1. Traffic flow information (Figure 1a), step counter calculating number of steps walked and number of calories burned (Figure 1b) and an accident detection system where the user can insert an emergency number and the phone can auto detect in case an accident happens and send emergency SMS to that number. An accident is basically detected via aggressive and abnormal vibration behaviour of the mobile phone.



Steps: 9
Distance: 0.007 km
Calories: 0.45

Steps: 9
Distance: 0.007 km
Calories: 0.45

Send a SMS to a contact when an accident is detected

There is no phone saved. Be sure to select a contact with phone number

(a) Traffic view (b) Step counter view (c) Notify accident view

Figure 1. MobilitApp current services

During this phase of MobilitApp project, I will work to add the functionality of detection for the means of transportation the phone-holder is using.

3 Project Structure

3.1 Project Objective

As mentioned in the previous section, the initial objective of this thesis is to make MobilitApp capable of detecting the user's means of transportation, for example use of a bus, train, metro…etc. This activity detection algorithm should be on real-time basis.

The concept followed was based on machine learning technology where the following 2 stages should be fulfilled:

- Collecting data from mobile phone sensors;
- Creating a machine learning algorithm to detect the means of transportation.

This transportation activity detection logic is developed to serve its purpose in the city of Barcelona, Spain. It should be capable of detecting 10 transportation activities shown in Table 2.

Table 2. Transportation activities targeted to be detected by MobilitApp

Activity Name	**Activity Description**
Stationary	The device is not moving.
Walk	The device is on a user who is walking.
Run	The device is on a user who is running.
Bicycle	The device is on a bicycle.
Motorbike	The device is on a motorbike.
Car	The device is on a car.
Bus	The device is on a bus.
Tram	The device is on a tram.

Metro	The device is on a metro.
Train	The device is on a train.

Android Activity Recognition API mentioned in section 2.1.5 already can detect 8 user activities (IN_VEHICLE, ON_BICYCLE, ON_FOOT, RUNNING, STILL, TILTING, UNKNOWN, WALKING), but those activities do not include the differentiation between types of transportation means. Also, it includes an activity called “Unknown” which affects the resulting output badly whenever it is detected. Consequently, I added another point to the thesis objective which is considering the enhancement I should make in my activity detection algorithm versus that of Android Activity Recognition API.

3.2 Project Phases

In order to reach my target, the work was phased out into the 4 following phases and the scope of each of them will be discussed in chapter 4:

Phase 1: Data Capturing Logic Development;

Phase 2: Data Collection;

Phase 3: Activity Detection Logic Development;

Phase 4: Migrating Detection Logic to MobilitApp.

4 Development Process

4.1 Phase 1: Data Capturing Logic Development

First of all, a logic was developed on MobilitApp to capture and log the data from accelerometer, magnetometer and Android Activity Recognition API. I focused on accelerometer because it is the sensor responsible for motion detection in an Android phone, as per Android developers [10]. Also because of its wide footprint, now, you can barely find a smartphone without accelerometer. Magnetometer comes in the second place just as an assistant for the accelerometer to enhance its detection. Moreover, I triggered the data of Android Activity Recognition API to illustrate the difference between its output and output of MobilitApp developed logic.

The data will first be logged in the phone's storage, then it should be upload to a Raspberry PI server where it will be analysed.

Enhancement was done to the user Interface (UI) of MobilitApp where activity annotation functionality was added to assist in data collection, as show in Figure 2.

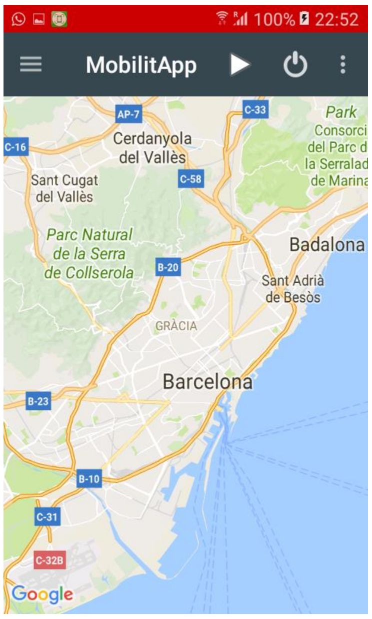
a) The phone user should press the play button to start recording the data of the transportation activity being done. See Figure 2 (a).

b) Select the type of transportation activity from a pop-up menu. See Figure 2 (b).

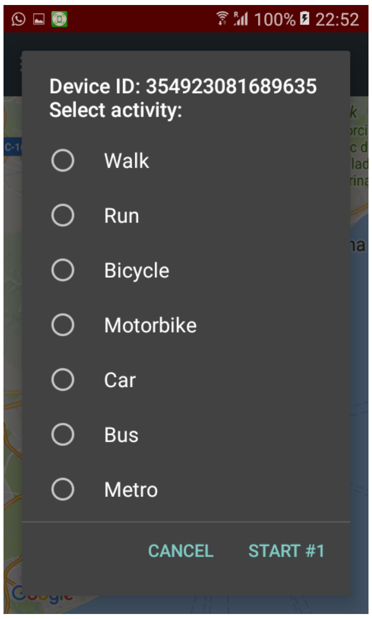
c) Press the stop button after finishing the transportation activity to stop the data recording. See Figure 2 (c).

d) Discard the logged data or save it to the phone's storage. See Figure 2 (d). This is because in some cases the users forget to stop the recording after the transportation activity has been finished. So if the data is saved and sent to the server it will be misleading.

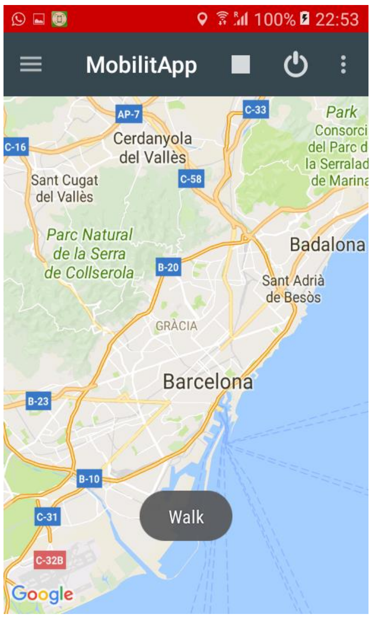
e) Press the "Send data" choice from the dropdown menu to send the logged data files to the server and delete it from the mobile storage. See Figure 2 (e).



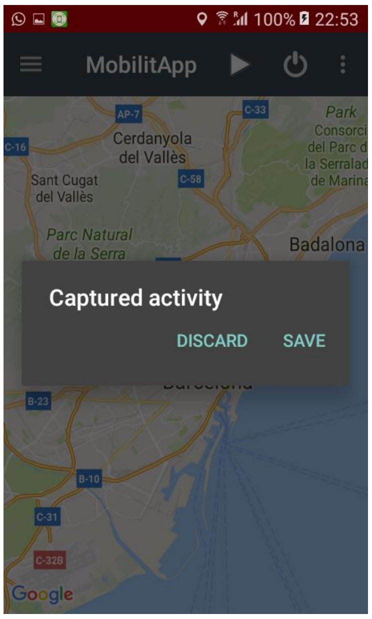
(a) Start recording


Device ID: 354923081689635
Select activity:

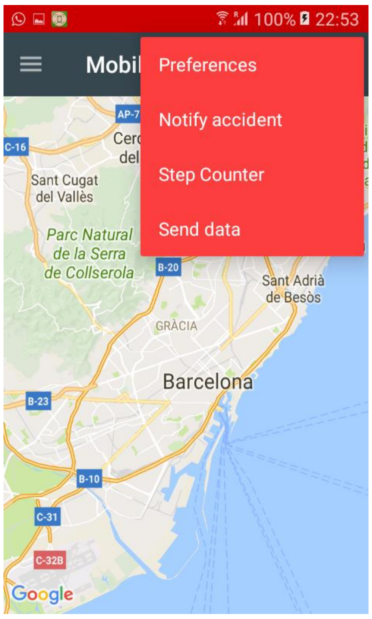
(b) Choose activity type



(c) Stop recording


Captured activity

(d) Save the recorded sample



(e) Upload the recorded data to the server

Figure 2. Transportation activity data recording UI

4.2 Phase 2: Data Collection

In this phase, I focused on acquiring volunteers to assist us in collecting data by using MobilitApp via the new module developed in phase 1. First of all, slides were prepared explaining how to use MobilitApp data collection module [11]. Then, a campaign was launched to acquire volunteers to use the app and collect test data volunteer acquiring campaigns were launched via:

- Emails & WhatsApp: emails and WhatsApp messages were formulated and sent to groups of professors and students of UPC.
- MobilitApp webpage [9]: New section was added to the site to esteem the site visitors to assist us in collecting data.

During this phase, a fierce challenge was faced in convincing people to volunteer for using the app to collect data for the sake of development. Eventually, I managed to collect data from 8 mobile devices covering all the 10 required transportation activities. Table 3 shows the duration of the data collected per activity.

Table 3. Collected duration per activity

Activity	**Overall Collected Duration of Recordings, [min]**
Car	25
Bicycle	38
Train	44
Tram	46
Motorbike	46
Run	48
Stationary	62
Metro	251
Bus	262
Walk	417

4.3 Phase 3: Activity Detection Logic Development

After collecting the needed data, a machine learning model needed to be set in order to detect the transportation activities. In the next section, machine learning concepts and models are illustrated to consequently conclude the appropriate machine learning model needed in our case.

4.3.1 Machine learning concept assessment

Machine learning is divided into 2 concepts as summarized in Figure 3: [12] [13] [14]

4.3.1.1 Supervised machine learning

Supervised machine learning is based on building a model from datasets with known responses (labels) known as training data and the resulting model is known as trained model. Consequently, this trained model can be used to predict responses (labels) for other datasets with unknown responses.

Supervised learning common techniques are classification and regression.

Classification is used in case of prediction of discrete responses like in questions with an answer of yes or no. For instance, whether an email is genuine or spam. Classification models classify input data into categories.

Classification common algorithms are:

- Support vector machine (SVM)
- Boosted and bagged decision trees
- K-nearest neighbour, Naïve Bayes
- Discriminant analysis, logistic regression
- Neural networks

Regression is used in case of prediction of continuous responses. Here, the answer of the question is not in a discrete format, it always varies with a range which is unlimited in most cases. For instance, temperature changes or employees' salaries predictions.

Common regression algorithms are:

- Linear model
- Nonlinear model
- Regularization
- Stepwise regression
- Boosted and bagged decision trees
- Neural networks

4.3.1.2 Unsupervised machine learning

Unsupervised learning is based on building a model from unlabelled datasets. The training data here has no known responses (labels). Unsupervised learning technique works on finding the hidden patterns and drawing inferences for the unlabelled training data.

Clustering is the most common unsupervised learning technique. It works on forming clusters or grouping the data based on the hidden patterns. It can be used in market research or for customers' segmentation.

Common clustering algorithms are:

- K-means and k-medoids
- Hierarchical clustering
- Gaussian mixture models
- Hidden Markov models
- Self-organizing maps

- Fuzzy c-means clustering
- Subtractive clustering

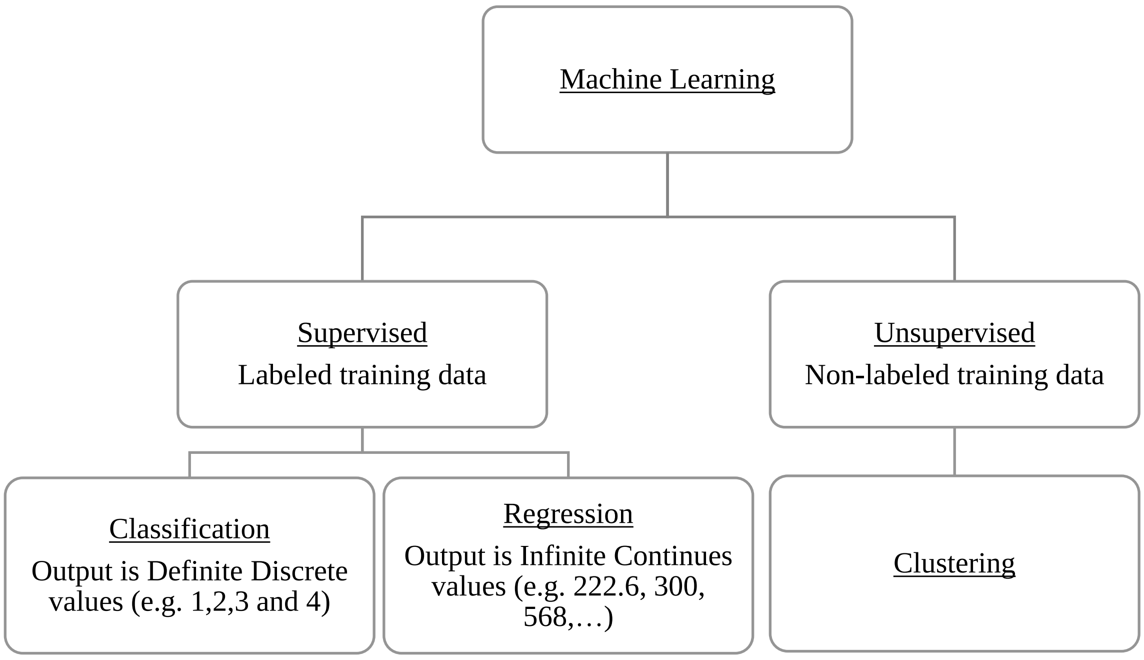

Machine Learning
Supervised
Labeled training data
Unsupervised
Non-labeled training data
Classification
Output is Definite Discrete values (e.g. 1,2,3 and 4)
Regression
Output is Infinite Continues values (e.g. 222.6, 300, 568,…)
Clustering

Figure 3. Machine learning concepts

In our case, Regression Supervised machine learning is the best fitting concept because data which we will use to train the model is labelled meaning that we know the activity accompanying each data input. That is why I will use supervised way and I choose regression because the data is an output of accelerometer and magnetometer so the data form is continuous not discrete values meaning that it can have any value with no limits.

4.3.2 Data structure assessment

In this part, I will focus on the accelerometer data. The initial data which I got from the accelerometer was somehow random but I can categorize some of the 10 activities to be classified in groups. The stationary activity components always have a very tiny variation in the magnitudes of the X, Y, Z components (most of the time, the variation of magnitude is less than 0.1 between certain instance and its successive one). Run and walk activities tend to have a uniform patterns which are being repeated over and over. Other activities are very random which make it difficult for the human eyes to distinguish between them.

As shown in Figures 4 – 13. For this reason, machine learning was the best choice to go for to assist me in distinguishing between the different activities. This was the main scope machine learning was invented for, to detect what the human-being cannot detect by normal static coding.

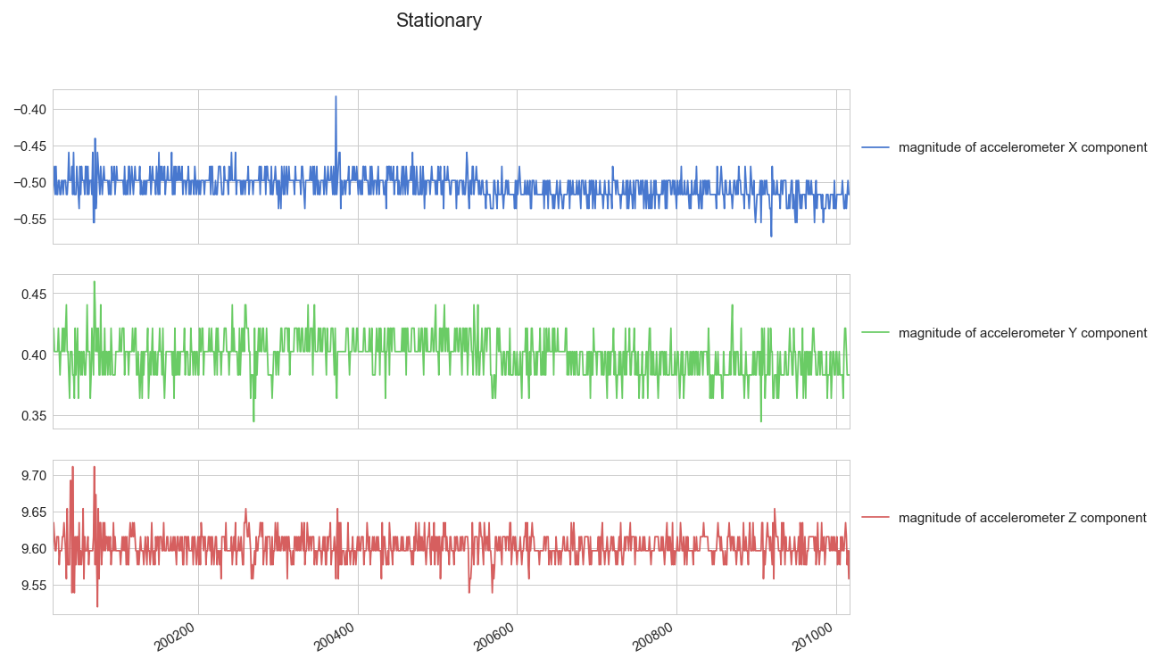


Figure 4. Stationary activity magnitudes of accelerometer components (m/s^2) for ~6 s

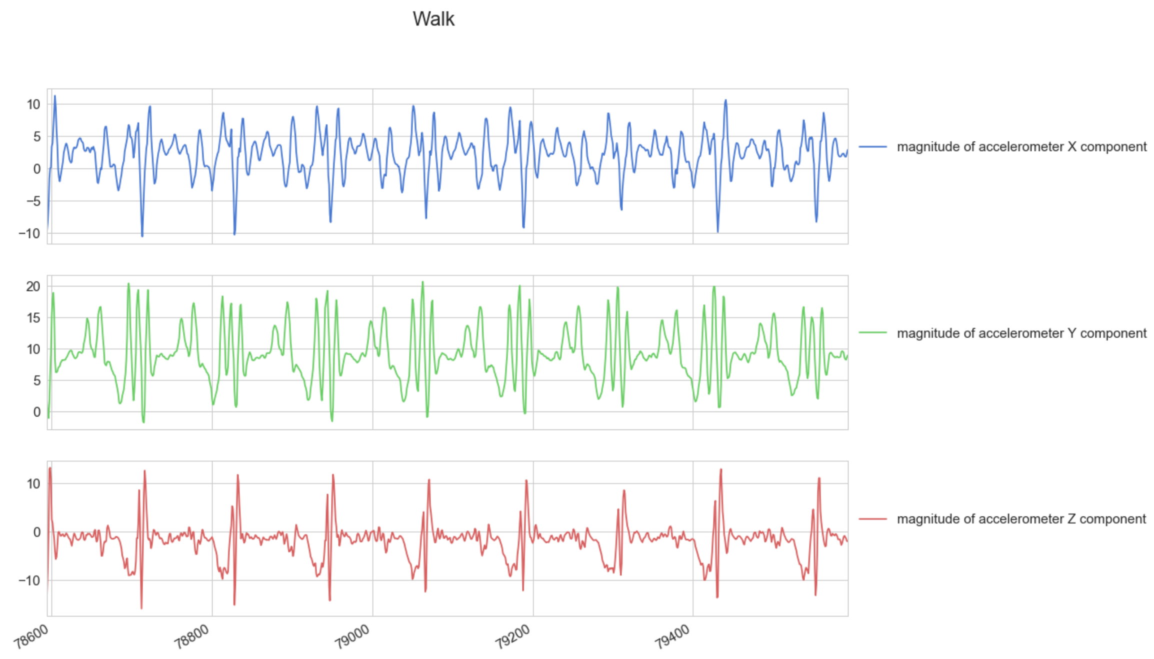


Figure 5 Walk activity magnitudes of accelerometer components (m/s^2) for ~6 s

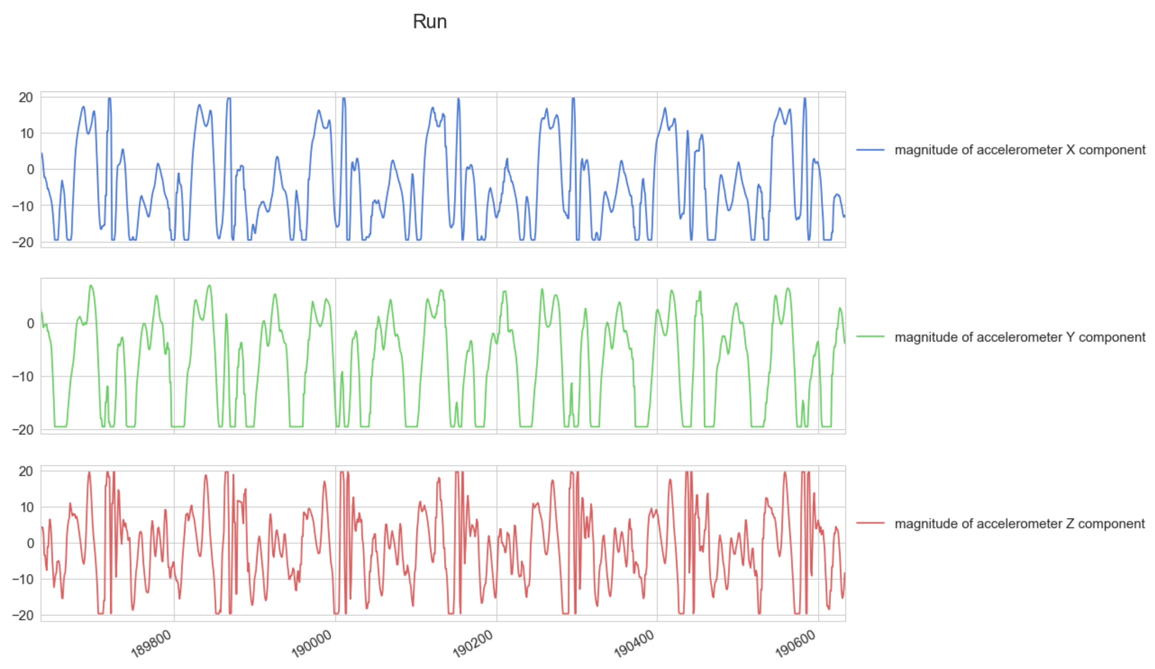


Figure 6 Run activity magnitudes of accelerometer components (m/s^2) for ~6 s

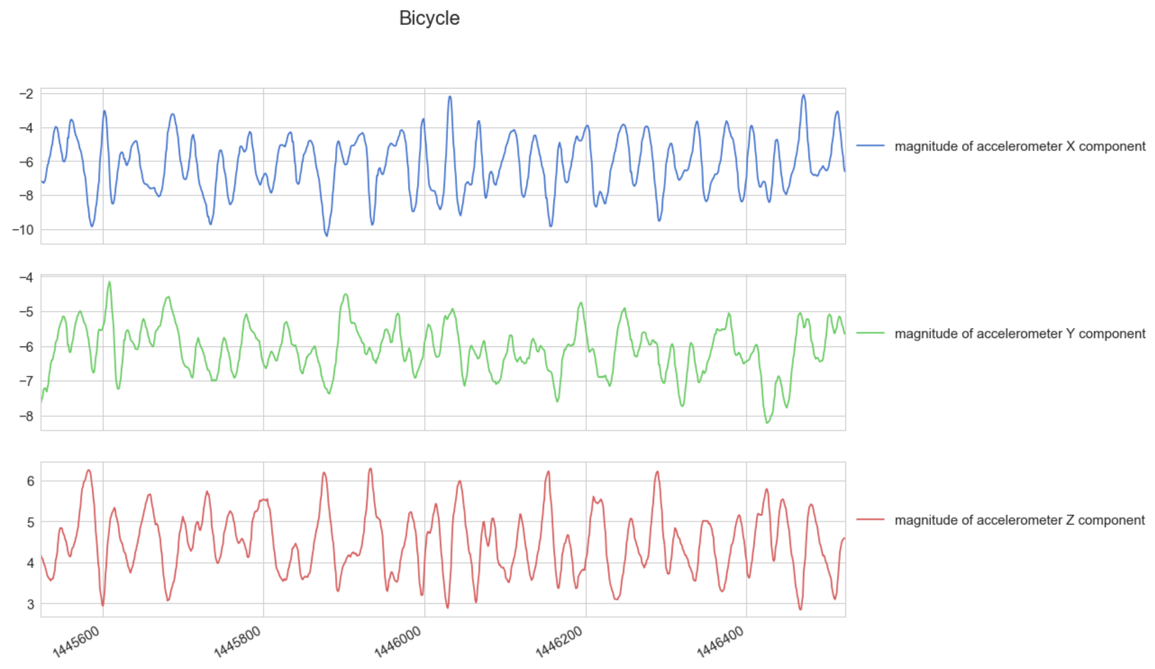


Figure 7 Bicycle activity magnitudes of accelerometer components (m/s^2) for ~6 s

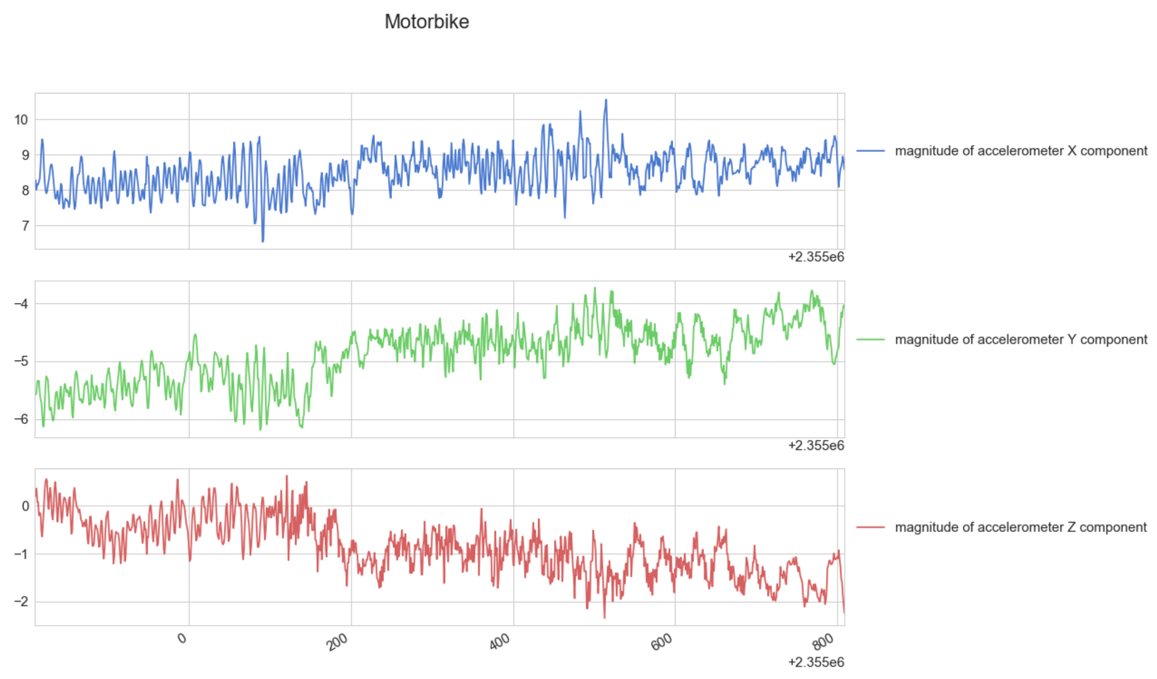


Figure 8 Motorbike activity magnitudes of accelerometer components (m/s^2) for ~6 s

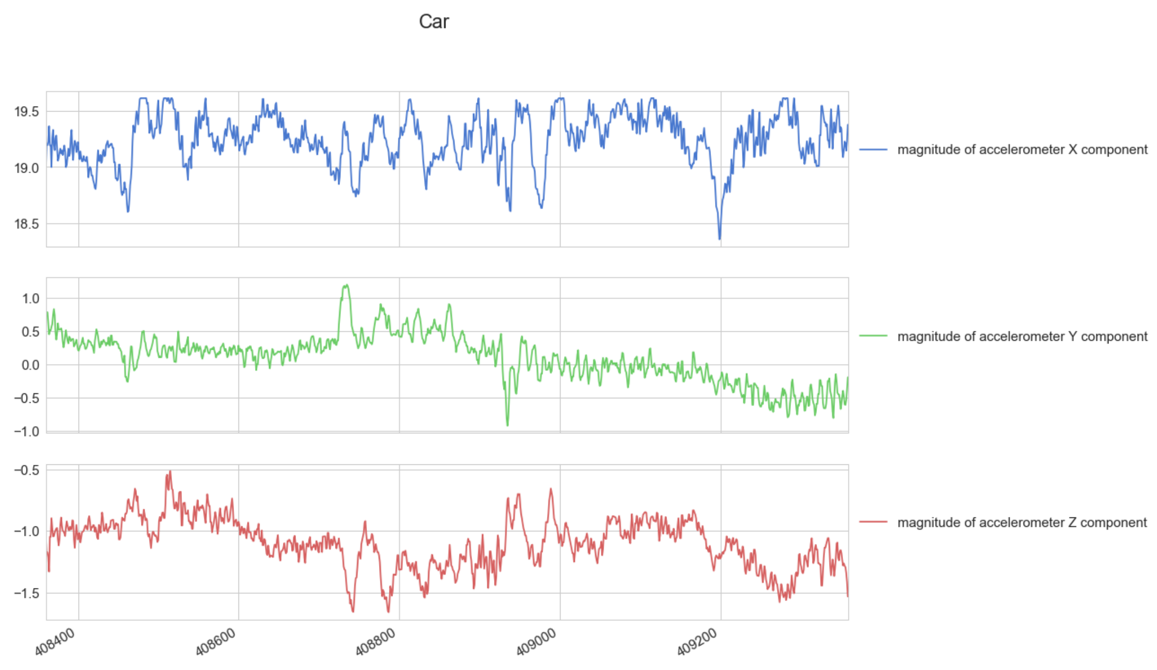


Figure 9 Car activity magnitudes of accelerometer components (m/s^2) for ~6 s

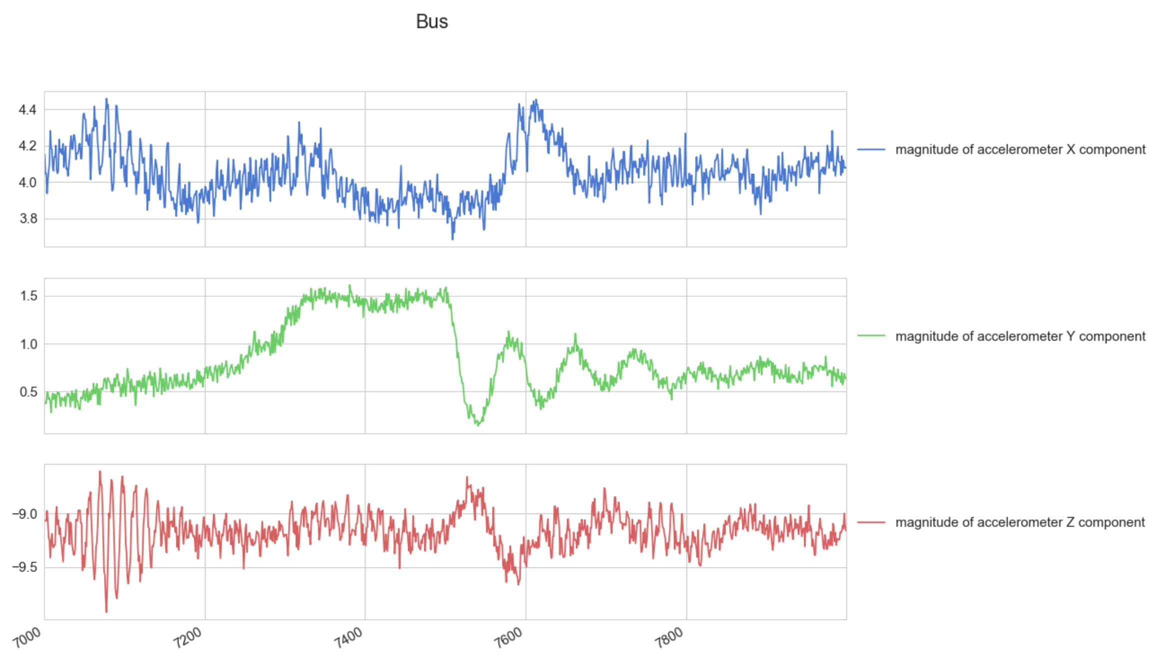


Figure 10 Bus activity magnitudes of accelerometer components (m/s^2) for ~6 s

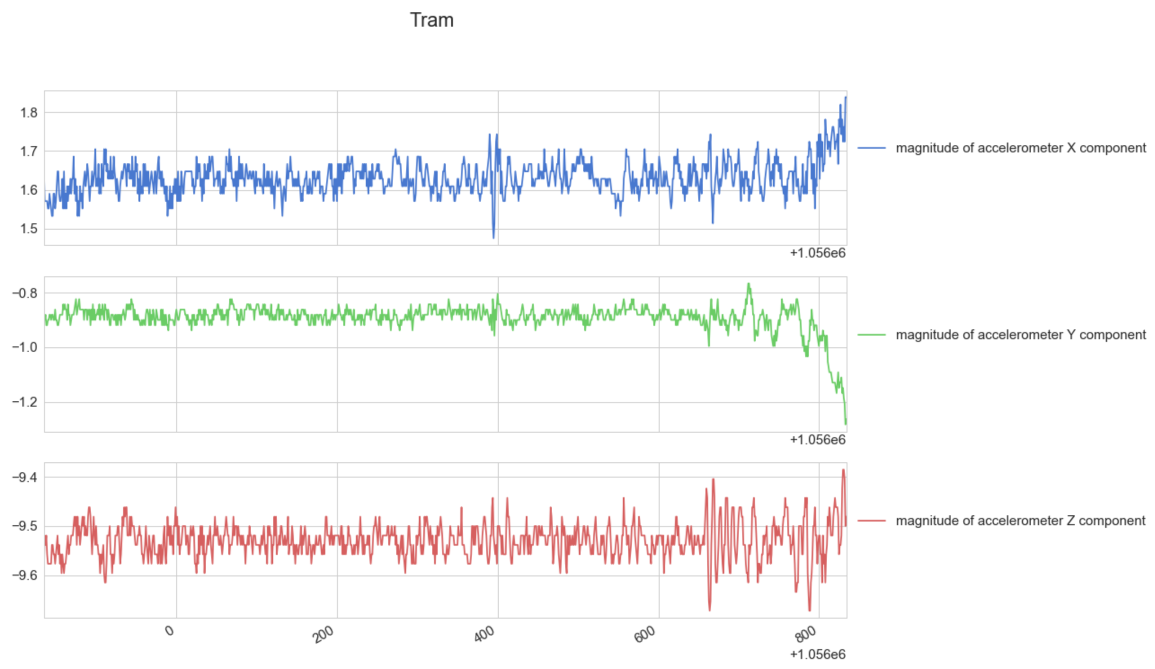


Figure 11 Tram activity magnitudes of accelerometer components (m/s^2) for ~6 s

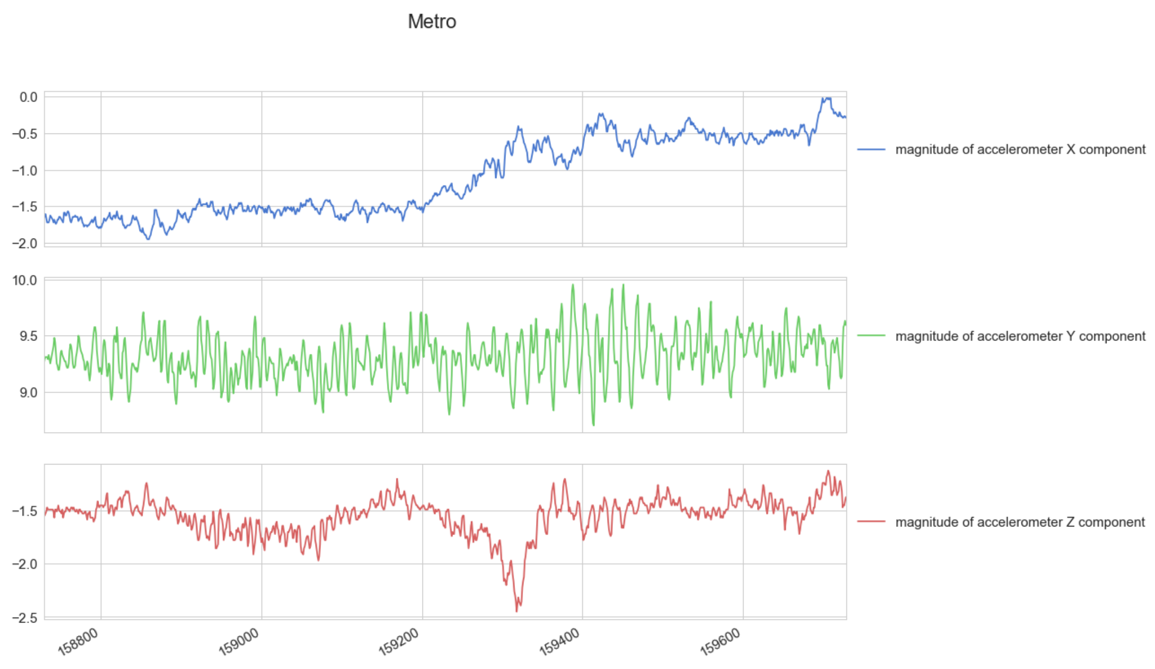


Figure 12 Metro activity magnitudes of accelerometer components (m/s^2) for ~6 s

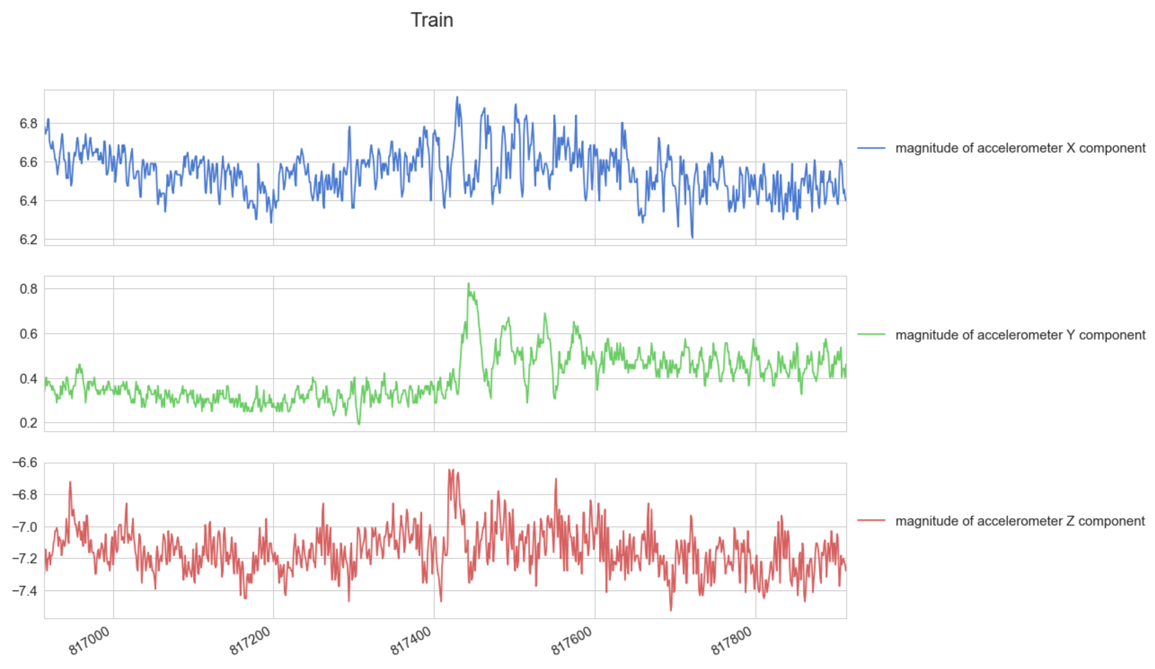


Figure 13 Train activity magnitudes of accelerometer components (m/s^2) for ~6 s

For machine learning, I needed to restructure the data to fit for the machine learning concepts. First of all the machine learning model should be fed the data as intervals to be able to deduce the pattern of the movement. Because giving the model just 3 points in time for each input was meaningless. The second challenge was to decide what parameters these data should form and fed the machine learning model. As per my research and multiple trials, the best data structure was to calculate the mean, the standard deviation and the first component of the Principal Component Analysis (PCA) for each component of the accelerometer (X, Y, and Z), see Figure 14. The same parameters were calculated for the magnetometer. Those parameters were calculated for each successive 1000 log of each component representing a time interval of average 6 ms which is good for running the application on real-time basis considering that MobilitApp will predict the transportation activity each 6 ms. Finally, I got the required parameters for each activity to train the machine learning model to be able to detect the activity.

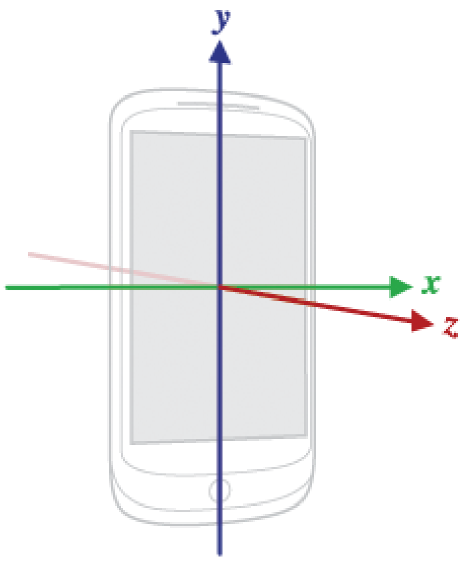


Figure 14 Android sensors components (accelerometer and magnetometer) [15]

4.3.3 Machine learning model assessment

In this stage, I used Matlab software [16] as our tool to assess the machine learning models versus our training data structure. Matlab was the best option for this stage because it provides us with a classification learner application with which multiple models can be trained and compare between their performance. Trial and error played a big role in finding the right model to work with, but also formatting the data in an appropriate structure for the model was another challenge. The best performing machine learning models, as expected, were those of ensemble methodologies [17] because they adopt the concept of training multiple models using the same learning algorithm which as a result enhance the prediction ability of the trained model. The two major ensemble models are Bagging and Boosting trees [18]. Both models work on training multiple learners and then conclude the final trained model. I used 30 learners for our runs which is the default value for Matlab. The major difference is that bagging get the simple average from the trained learners to conclude the model while boosting give weights to each learner where poor ones get lower weights to enhance its final output. This is shown in Figure 15.

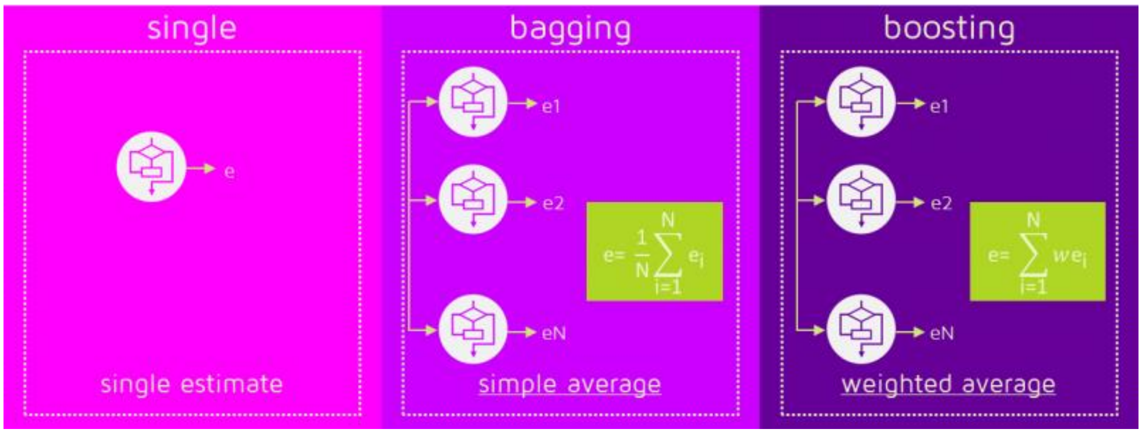


Figure 15 Bagging versus boosting classifiers' training [18]

To compare between trained models in the training stage, I used the holdout validation method, where the data is split into 2 portions. 75% of the data is used for training and 25% of the data is used to test the resulting trained model. Table 4 shows the results of the holdout validation where ensemble bagged and boosted trees shows the best performance. But the best performer was always ensemble bagged trees through all of our trial sets. I also trained the other models to keep them as a reference for the resulted models performance.

Table 4. Holdout validation results for machine learning models

Trained Model	**Holdout Validation Accuracy**
Ensemble Bagged Trees	90.20%
Ensemble Boosted Trees	83.00%
Medium Tree	75.50%
Ensemble RUSBoosted Trees	74.00%
Simple Tree	62.10%

Fine KNN	52.00%
Ensemble Subspace KNN	51.90%
Weighted KNN	51.80%
Cosine KNN	51.50%
Medium KNN	51.40%
Cubic KNN	51.30%
Coarse KNN	48.50%
Linear Discriminant	47.10%
Ensemble Subspace Discriminant	46.60%
Cubic SVM	35.80%
Quadratic SVM	35.70%
Medium Gaussian SVM	35.50%
Linear SVM	33.90%
Coarse Gaussian SVM	33.00%
Fine Gaussian SVM	31.80%

In order to dig down in the resulted data, I will use the confusion matrix chart. The confusion matrix shows success percentages of the predicted activities by the model in its horizontal axis and the actual true activities in its vertical axis. This will give us insights not only on the overall performance of the trained model, but also on its performance in detecting each activity.

As shown in Figure 16, the maximum success rate was of 98% achieved on detecting the bicycle activity and the minimum success rate was of 61% achieved on detecting the car activity.

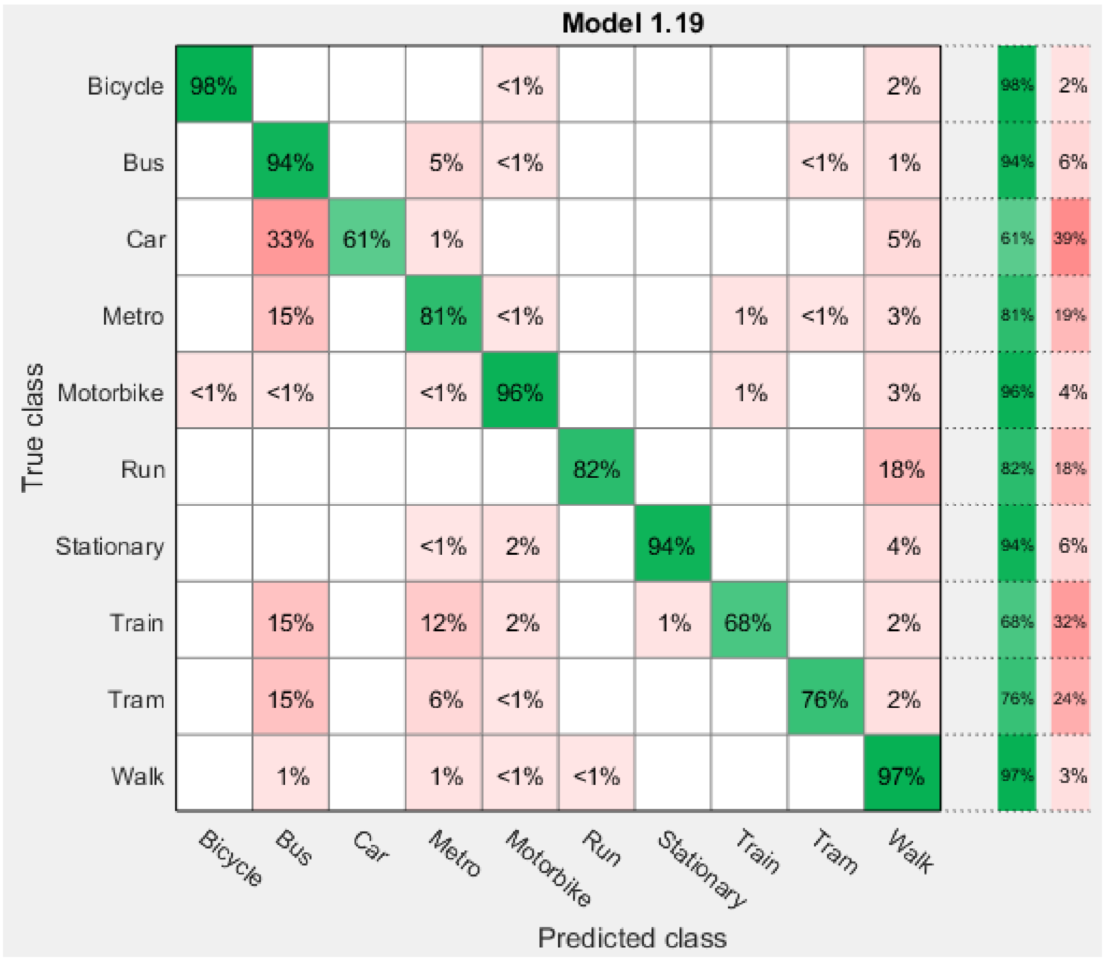


Figure 16 Confusion matrix showing results of testing data on the trained model

<u>Performance Comparison with Android Activity Recognition API</u>

I used multiple sets of live data to resemble the live performance of the Ensemble Bagged Trees trained model. The average success rate was around 77%, while the average success rate of Android Activity Recognition API was 50% that is significantly lower.

As Activity Recognition API do not cover all the 10 activities targeted by MobilitApp logic I considered the mapping in Table 5.

Table 5 Mapping Google Activity Recognition API activities to MobilitApp activities

Activity Recognition Activity	**MobilitApp Activity**
IN_VEHICLE	Bus, Car, Metro, Train, Tram

ON_BICYCLE	Bicycle
ON_FOOT	Walk
RUNNING	Run
STILL	Stationary
TILTING	-
UNKNOWN	-
WALKING	Walk

In Table 6, I choose the common detected activities among Android Activity recognition API and my MobilitApp logic to give an insight over the performance of detection of each of them. Android Activity API shows lower performance, but the obvious case was in the "Run" activity, as it gave 0% success rate. This is because actually the people who were doing those running activities were jogging not actually running. Thus, my resolution for that is that Android Activity Recognition API needs a very defined activity shape to be able to detect it.

Table 6 Android Activity Recognition API versus MobilitApp

Activity Name	**Android Activity Recognition API Success Rate**	**MobilitApp Success Rate**
Bicycle	43%	96%
Stationary	25%	80%
Walk	73%	84%
Run	0%	77%

4.4 Phase 4: Migrating Activity Detection Logic to MobilitApp

In this phase, efforts were done to migrate the machine learning model to Android over MobilitApp application. In order to do this multiple tools were assessed to settle down on the optimal one. In the following, I will show those different tools from the point of view of Android compatibility and results efficiency in our case:

4.4.1 Matlab [16]

- Android Compatibility: Not compatible
- Results Efficiency: High

Matlab was the first candidate to integrate the machine learning model with Java [19] which is used in MobilitApp development and currently used to develop most of Android apps. The 2 options to integrate Matlab code in Java was whether to convert it to C language [20] or to create a stand-alone Java application. The obstacle here was that both methods cannot convert the machine learning classifier, produced from training the machine learning model, to C or a stand-alone Java application. Consequently, it is not feasible to directly deal with Matlab in this phase because code is only dedicated to work in Matlab.

4.4.2 Weka [21] [22]

- Android Compatibility: Compatible
- Results Efficiency: Low

Weka (Waikato Environment for Knowledge Analysis) is a tool which avails multiple machine learning methodologies. It is written in Java and developed at the University of Waikato, New Zealand. It is free software licensed under the GNU General Public License. The obstacle with Weka was that the machine learning models trained by Weka did not give predictions of high efficiency as that of Matlab, but the major defect was that there are whole activities cannot be detected at all like Tram and Train.

4.4.3 Tensorflow [23] [24]

- Android Compatibility: Compatible
- Results Efficiency: Low

TensorFlow is a library which can be used for machine learning applications. It is an open-source software library developed by Google Brain Team. When I used TensorFlow in this phase it gave 95% success rate during the training phase, but on live data, it gave a very low efficient result of average success rate of 10%. Accordingly, I excluded moving forward with TensorFlow.

4.4.4 Python [25] [26] [27]

Python is a high-level programming language. Python is the strongest programming language with machine learning. It has a robust machine learning libraries. First, I implemented the machine learning model using Python over personal computer (PC) and it gave an average success rate of 68% during training and 70% during testing its trained model with live data. Results gained from Matlab, previously in section 4.3.3, were 90.2% during training and 77% during testing with live data. Although, testing data validation for Matlab was better, but I considered Python because in training and testing phases it nearly gave results as Matlab with live data. Second, I needed to find a way to integrate Python with its machine learning libraries over Android. There are a lot of solutions developed to integrate Python over Andoid's Java. The most important is to find the solution which avails the integration of Python's machine learning libraries as well. The current available solutions are divided into 2 groups. The first group with target to build a standalone APK over Android and this can assist me in integration of Python with Java. The second group is enabling a terminal over Android in which we can write Python code over Android and this group can be used just to proof that Python utilities can be used over Android. In the following, I will show those solutions from 2 viewpoints. As shown in Table 7. The first viewpoint is the feasibility for the solution to be integrated with Java and I will put it under the name of "overall integration with Java". The second viewpoint is if the solution avails the libraries required to use Python in machine learning and I will put it under the name of "Python machine learning libraries integration with Java". In our case, in order to use Python in machine learning, the following 3 libraries are needed to be imported to the Python code:

- Pandas [28]: I used it in getting data from files and restructure it.
- Pickle [29]: I used it in storing and loading the trained machine learning model.

- Scikit-learn [30]: I used it to import the required machine learning algorithm to work with.

4.4.4.1 Build Standalone APK

As shown in Table 7, as per my assessment for the shown solutions, all of them are not feasible in case of integration of Python Machine Learning libraries, except for Chaquopy [31]. Chaquopy's team worked on integrating the required libraries (pandas, scikit-learn) during the time of writing this thesis and they managed to successfully launch the new version in 26.04.2018. Thus, my recommendation was to stick to Chaquopy in this phase. I need to highlight that Chaquopy is not an open source SDK and it costs 39 Euro/developer.

Table 7. APK integration solutions of Python over Android

Solution	**Overall integration with Java**	**Python Machine Learning libraries integration with Java**
BeeWare tool (https://pybee.org/)	Feasible	Not feasible
Chaquopy SDK (https://chaquo.com/)	Feasible	Feasible (26.04.2018)
Kivy library (https://kivy.org/)	Not feasible	Not feasible
python-for-android tool (http://python-for-android.readthedocs.io/)	Not feasible	Not feasible

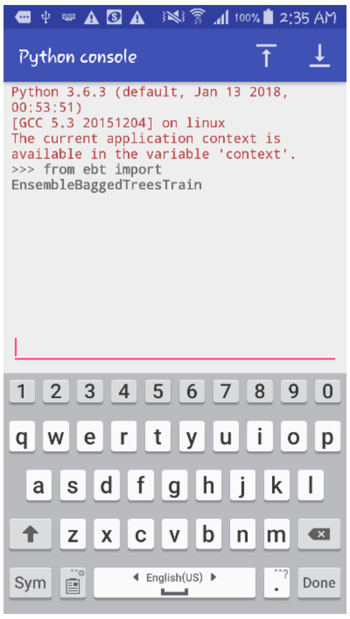
4.4.4.2 Android Terminal

There are 2 terminals which can be downloaded from Google Play like any normal application and after installing them, they can be used to write Python code. This method cannot assist in integrating my solution in MobilitApp, but can be used to proof the concept that machine learning using Python can function over Android OS. As shown in Table 8, Termux is the one which supports working with Python Machine Learning libraries. Qpython, however in their website they stated that it can handle machine learning libraries. In my live testing to it, its performance was very poor, not consistent and not all functions were working properly.

Table 8 Terminal solutions of Python over Android

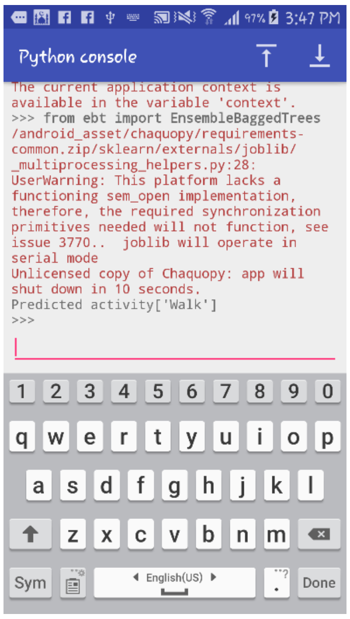
Solution	**Overall integration with Java**	**Python Machine Learning libraries**
Qpython (http://www.qpython.com/)	Not feasible	Not Feasible
Termux (https://termux.com/)	Not feasible	Feasible

In conclusion, the best option, as I mentioned before, is to go for Chaquopy SDK, as it can integrate Python with Android applications built over Java. I have managed to perform a proof of concept using Chaquopy unlicensed version. As shown in Figure 17, via Chaquopy, I managed to run my Python script over Android with the required libraries by which I was able to train Ensemble Bagged Trees model and use it to detect the transportation activities but not on real time basis.



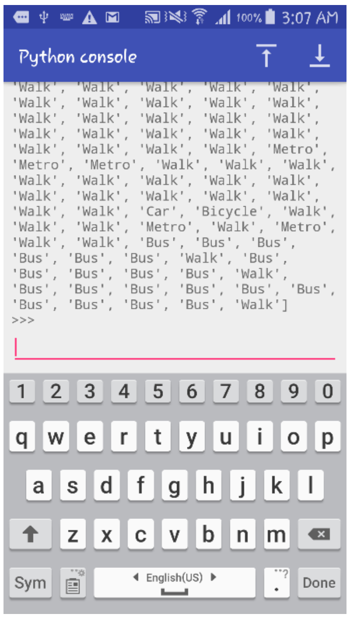
```
Python 3.6.3 (default, Jan 13 2018,
00:53:51)
[GCC 5.3 20151204] on linux
The current application context is
available in the variable 'context'.
>>> from ebt import
EnsembleBaggedTreesTrain
```

(a) Training model



```
The current application context is
available in the variable 'context'.
>>> from ebt import EnsembleBaggedTrees
/android_asset/chaquopy/requirements-
common.zip/sklearn/externals/joblib/
_multiprocessing_helpers.py:28:
UserWarning: This platform lacks a
functioning sem_open implementation,
therefore, the required synchronization
primitives needed will not function, see
issue 3770..  joblib will operate in
serial mode
Unlicensed copy of Chaquopy: app will
shut down in 10 seconds.
Predicted activity['Walk']
>>>
```

(b) Detect a single activity (walk)



```
'Walk', 'Walk', 'Walk', 'Walk', 'Walk',
'Walk', 'Walk', 'Walk', 'Walk', 'Walk',
'Walk', 'Walk', 'Walk', 'Walk', 'Walk',
'Walk', 'Walk', 'Walk', 'Walk', 'Walk',
'Walk', 'Walk', 'Walk', 'Walk', 'Metro',
'Metro', 'Metro', 'Walk', 'Walk', 'Walk',
'Walk', 'Walk', 'Walk', 'Walk', 'Walk',
'Walk', 'Walk', 'Walk', 'Walk', 'Walk',
'Walk', 'Walk', 'Car', 'Bicycle', 'Walk',
'Walk', 'Walk', 'Metro', 'Walk', 'Metro',
'Walk', 'Walk', 'Bus', 'Bus', 'Bus',
'Bus', 'Bus', 'Bus', 'Walk', 'Bus',
'Bus', 'Bus', 'Bus', 'Bus', 'Walk',
'Bus', 'Bus', 'Bus', 'Bus', 'Bus', 'Bus',
'Bus', 'Bus', 'Bus', 'Bus', 'Walk']
>>>
```

(c) Detect multiple activities

Figure 17 Chaquopy [32] [33] over Android OS

5 Results and Conclusion

My main target was to add the functionality of transportation activity detection in MobilitApp smartphone application.

I managed to reach an algorithm via machine learning Ensemble Bagged trees model to predict the different transportation activities in the city of Barcelona. The algorithm can detect between 10 transportation activities with average success rate of 77% (Stationary, Walk, Run, Bicycle, Motorbike, Car, Bus, Metro, Train and Tram). Compared to Android Activity Recognition API, 6 more motion activities can be recognized and the average recognition success rate is 1.5 times higher.

Currently, I proved my concept via running it using Matlab, Python over PC and Android OS.

<u>Future Work</u>

For the next phase, it is needed to integrate the transportation activity detection functionality in MobilitApp. This can be done by importing trained Ensemble Bagged Trees model in Android by using Chaquopy SDK.

For production and deployment of MobilitApp for commercial use, massive data should be collected from variant parts of Barcelona and retrain the machine learning model to enhance its detection efficiency over larger scale. Moreover, a dashboard should be developed with analysable views for business users, in order to make the output data beneficial and can be used to enhance the transportation infrastructure over the city.

References

[1] UPC, “MobilitApp,” [Online]. Available: http://mobilitat.upc.edu/.

[2] “UPC,” [Online]. Available: https://www.upc.edu/ca.

[3] Google, “Android Activity Recognition API,” [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionClient.

[4] “Moovit,” [Online]. Available: https://moovit.com/.

[5] “Google Maps,” [Online]. Available: https://www.google.com/maps.

[6] “My Smart Route,” [Online]. Available: http://mysmartroute.com/.

[7] “Waze,” [Online]. Available: https://www.waze.com/.

[8] “By2ollak,” [Online]. Available: https://desktop.bey2ollak.com/.

[9] “MobilitApp,” [Online]. Available: http://mobilitat.upc.edu/.

[10] “https://developer.android.com/guide/topics/sensors/sensors_overview.html,” Google. [Online].

[11] “MobilitApp Data Collection,” [Online]. Available: http://mobilitat.upc.edu/data_collection_steps.pdf.

[12] Mathworks, “Mathworks,” [Online]. Available: https://www.mathworks.com/discovery/machine-learning.html.

[13] L. L. Cárdenas, “Enhancement of vehicular ad hoc networks using machine learning techniques and privacy,” Leticia Lemus Cárdenas, Barcelona, 2018.

[14] “azure ML,” [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice.

[15] “Android sensor components,” [Online]. Available: https://github.com/imthexie/GestureLearner/wiki/Android-Accelerometer.

[16] “Matlab,” [Online]. Available: https://www.mathworks.com/products/matlab.html.

[17] D. O. &. R. Maclin, “Popular Ensemble Methods: An Empirical Study,” [Online]. Available: https://arxiv.org/pdf/1106.0257.pdf.

[18] “What is the difference between Bagging and Boosting?,” [Online]. Available: https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/.

[19] “Java for Android,” [Online]. Available: https://developer.android.com/studio/write/java8-support.

[20] “C language,” [Online]. Available: https://en.wikipedia.org/wiki/C_(programming_language).

[21] “Weka,” [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/.

[22] “Wikipedia (Weka),” [Online]. Available: https://en.wikipedia.org/wiki/Weka_(machine_learning).

[23] “TensorFlow,” [Online]. Available: https://www.tensorflow.org/.

[24] "Wikipedia (TensorFlow)," [Online]. Available: https://en.wikipedia.org/wiki/TensorFlow.

[25] "Python," [Online]. Available: https://www.python.org/.

[26] "Wikipedia (Python)," [Online]. Available: https://en.wikipedia.org/wiki/Python_(programming_language).

[27] "Python for Android," [Online]. Available: https://wiki.python.org/moin/Android.

[28] "Python Pandas," [Online]. Available: https://pandas.pydata.org/.

[29] "Python Pickle," [Online]. Available: https://docs.python.org/3/library/pickle.html.

[30] "Python Scikit-learn," [Online]. Available: http://scikit-learn.org/stable/.

[31] "Chaquopy," [Online]. Available: https://chaquo.com/chaquopy/.

[32] "Chaquopy," [Online]. Available: https://chaquo.com/chaquopy/.

[33] "Chaquopy Demo," [Online]. Available: https://github.com/chaquo/chaquopy.

[34] M. Aguilar, "Monica Aguilar," [Online]. Available: http://www-entel.upc.edu/monica.aguilar/.

[35] Wikipedia, "PCA," [Online]. Available: https://en.wikipedia.org/wiki/Principal_component_analysis.

[36] "Android Studio," [Online]. Available: https://developer.android.com/studio/.

[37] G. M. Torregrosa, "Improvement of algorithms to identify transportation modes for MobilitApp, an Android Application to anonymously track citizens in Barcelona," Universitat Politècnica de Catalunya, Barcelona, 2016.

[38] A. S. G. A. V. D. B. a. G. N. Yan Michalevsky, "PowerSpy: Location Tracking using Mobile Device Power Analysis," Stanford.

[39] P. N. S. T. Samuli Hemminki, "Accelerometer-Based Transportation Mode," Helsinki.

[40] M. M. J. B. D. E. SASANK REDDY, "Using Mobile Phones to Determine Transportation Modes," Los Angeles.