TÉCNICO LISBOA

DetectBiklio - detect bicycle usage with an Android smartphone app

Bruno Alexandre Oliveira Santos

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Paulo Jorge Pires Ferreira

Examination Committee

Chairperson: Prof. Pedro Tiago Gonçalves Monteiro
Supervisor: Prof. Paulo Jorge Pires Ferreira
Member of the Committee: Prof. Luís Manuel Antunes Veiga

July 2021

Acknowledgments

Firstly, I would like to thank my parents for making this work possible. Thank them for their encouragement and caring, for always being there for me over all these years. Without their support, none of this would be happening.

I would also like to thank my grandparents and brothers for their encouragement and support throughout my college education.

I would also like to acknowledge my dissertation supervisor Prof. Paulo Jorge Pires Ferreira for his insight, support, time, and sharing of knowledge that has made this Thesis possible.

Last but not least, to all my friends and colleagues over these years that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.

To each and every one of you – Thank you.

Abstract

The increasing concern related to air pollution levels caused by motorized vehicles and obesity levels, along with the technology evolution and its decreasing cost (making it more affordable), created new strategies to reduce the emission and obesity levels. One of them is using the existing sensing power available on users smartphones and determine his transportation mode rewarding her/him with gifts if she/he chooses bicycle as his transportation mode. Thus, the goal of this work is to improve the existing transportation detection algorithm accuracy of Biklio, a sustainable mobility encouraging app. We propose a machine learning (ML) algorithm for detecting if a user is using a bicycle. The ML algorithm is based on Random Forest. It replaces the current solution used in Biklio that uses the Activity Recognition API from Google; increasing the detection accuracy, we expect to minimize the number of cases where the output does not match reality.

Keywords

Biklio, mobile phone sensors, hardware, software, machine learning, features, classification models.

Resumo

O aumento dos valores da poluição do ar causado pelos veículos motorizados e dos níveis de obesidade, juntamente com a evolução e consequente redução de custos da tecnologia (tornando-a mais acessível), deram origem a novas estratégias de reduzir os níveis de emissões e obesidade. Uma delas é usar os sensores existentes no telemóveis dos utilizadores de modo a determinar o seu meio de transporte, oferecendo recompensas aos utilizadores que escolham a bicicleta como o seu meio de transporte. Desta forma, o objetivo deste trabalho é melhorar a precisão do algoritmo de deteção do modo de transporte existente do Biklio, uma aplicação que encoraja a mobilidade sustentável. Propomos um algoritmo de machine learning (ML) para detetar se o utilizador está a andar de bicicleta. O algoritmo é baseado no Random Forest. Substitui o algoritmo actual do Biklio baseado na API da Google Activity Recognition; aumentando a sua precisão, esperamos minimizar o número de casos em que o resultado não coincide com a realidade.

Palavras Chave

Biklio, sensores, hardware, software, machine learning, features, modelos de classificação.

Contents

List of Figures

List of Tables

Listings

Acronyms

API	Application Program Interface
CPU	Central Processing Unit
GSM	Global System for Mobile Communications
OS	Operating System
UI	User Interface
GAR	Google Activity Recognition
RF	Random Forest
DT	Decision Tree
SVM	Support Vector Machine
ML	Machine Learning
GPS	Global Positioning System
GB	Gigabyte
ROM	Read Only Memory
RAM	Random Access Memory
CHMM	continuous Hidden Markov Model
DHMM	discrete Hidden Markov Model
CO2	Carbon Dioxide
OPX	OnePlus X
OP7	OnePlus 7
REDMI4	Xiaomi Redmi 4
ANN	Artificial neural network
CRF	Conditional random field
DBN	dynamic Bayesian network

BN	Bayesian network
WHO	World Health Organization
KNN	K-Nearest Neighbors
APs	Access Points
AP	Access Point
BT	Bluetooth
HMM	Hidden Markov Model
DNN	Deep Neural Network
MCU	Microcontroller Unit

1

Introduction

Contents

Nowadays, the increasing pollution levels are triggering more and more discussions around the world, substantially raising the communities concern once it can have a big negative impact on people's health and well-being in the close future. There are already plenty of diseases and several deaths caused by such pollution levels [3, 4]. There are numerous causes for the increase of such pollution being one of them the motorized vehicles [5, 6]. In fact, one of the biggest causes of pollution is the gases released by vehicles, being these used for most people to move around. Its excessive use, either for small distances, or the almost equal amount of vehicles and persons on the road, is concerning for the environment. The high car density is causing several problems around the world, not just the pollution they cause, but also the traffic and the nonexistence of enough parking spots. Many cities around Europe, are trying to fight against it by disallowing the circulation of most of the diesel vehicles in the next ten years [7] since these are currently the ones that pollute most. This has the objective of reducing car density, indirectly improving the other two aspects, traffic, and parking spots. The main goal of the motor industry is to replace most current vehicles with electric cars, an ecological and less environment harming solution.

Increasing as well, during the last decades, is obesity [8]. In fact, the World Health Organization (WHO) claims that worldwide obesity has nearly tripled since 1975. They also say that in 2016, more than 1.9 billion adults aged 18 years and older were overweight. Of these over 650 million adults were obese. Although obesity is not just increasing among adults. Children also have increased their obesity numbers, reaching 38 million, under the age of 5 in 2019, and 340 million children and adolescents aged 5-19 were overweight or obese in 2016. The principal cause of obesity is an imbalance of calories consumed and calories expended. Nowadays global diets have increased in the consumed calories, mainly energy-dense foods high in fat and free sugars, mostly caused by the fast-food market. On the other hand, the physical activity levels decreased due to the higher access to public transportations, or the nature of most works nowadays where the worker spends most of his day sited on a chair in front of a computer.

Alongside, there has been a fast and notable increase in the presence of smartphones in human beings daily life in the past years [9]. In fact, smartphones are becoming ubiquitous in nowadays society [10]. Everywhere we go, people are using their smartphones. That is noticeable looking at the continuous increase of devices per person nowadays [11]. This is due to the higher affordability of technology and its advance, allowing users to navigate on the web, send emails, play games, make calls, and many other actions in a simple smartphone. Smartphones are not only increasing in number but also computation, networking, and sensing power.

The combination of these three factors, smartphone presence, and its capabilities, along with the growing need to reduce the Carbon Dioxide (CO_2) emission levels, and the increasing obesity numbers, created a pretty interesting opportunity for the development of a new type of apps.

1.1 Motivation

There are several categories of apps, related to a healthy lifestyle, that use mobile phone sensors. We are going to consider three main categories: 1) apps that encourage sustainable mobility such as Biklio [12], (the one we are most interested in) or MatkaHupi [13]; 2) activity tracking apps for healthcare such as Activity [14]; 3) finally, activity-driven crowdsourcing apps such as Waze [15]. All these apps have in common that they are all based on activity recognition. Some of the available sensors on the smartphone [10] are used to predict an activity. All these apps also promote a healthier and environmentally friendly lifestyle encouraging CO2 emission reductions and the increase of physical activity.

In particular, Biklio is a sustainable mobility encouraging app that promotes the use of a greener travel mode, bicycle to be more specific. As an attempt to encourage people to use a bicycle, the app offers awards to users who choose to cycle, either to go to work, to the grocery store, or to tour around the city. The more a user rides a bike the more he/she earns. To be able to provide such awards to their users, the app has created some partnerships with local stores in the cities where the app is implemented and operating. Most shops give discounts when such users buy something after they have cycled. The app has already been tested in seven different countries, Portugal, Italy, Sweden, Luxembourg, Bulgaria, United Kingdom, and the Netherlands. The main intention here is to reduce the number of cars on the streets and consequently CO2 emissions.

To provide such offers to their users, the app determines if a user is cycling or not, and informs them of the award they can reclaim with a small time delay concerning the end of the cycling. Such a short delay is very important. In fact, users are allowed to reclaim their discounts as soon as they have finished the ride. It is also extremely important that the app detects if a user has effectively cycled; in fact, those who have not cycled should not be rewarded, and those who have should be able to reclaim the award. Thus, both the accuracy of the detection and the delay (as described above), are relevant for the Biklio app. Currently, the app relies on the sensors available in a user's smartphone and uses the Google Activity Recognition (GAR) Application Program Interface (API) [16] for that purpose.

1.2 Objectives and Challenges

The main objective of this work is to improve the existing cycling detection algorithm of the Biklio app [12]. Thus, we want to ensure that the best possible solution is implemented on the app, providing the most accurate results possible. For that, the following requirements need to be satisfied:

1. high accuracy (to ensure the closeness of the measurements to the real activity);
2. response time/delay must minimize the time needed to know the results;
3. battery usage must minimize the consumption caused by the app, and

4. software compatibility.

NOTE An android version is implemented to understand if the app algorithm improved the current solution.

This work has two main threads, a Machine Learning (ML) algorithm and the GAR. Both solutions allow meeting the goal following the given requirements. However, some challenges need to be taken into consideration. Some are related to both solutions but others are specific for only one of them.

Starting with the general, i.e., the ones that are applied for both solutions it is necessary to have in consideration the state of the sensors. If they are in good shape, working as they are supposed to, or returning no bizarre values. This is important to assure because, if the sensors are failing the solution can not be applied with the desired result. Also, the sensors model and brand can influence the readings because different sensors from different brands or even within the same brand but different models could return different values.

Following that arises another mutual concern, once the algorithm should work for every sensor or at least for the maximum number possible, to make it the more general possible. In other words, to guarantee that the algorithm works, no matter what device the user is using. This is important because each smartphone uses different sensors, not just in brands or models but also some have sensors that others do not.

Moving on to GAR API specific challenges the main one is the lack of control the programmer has. In this solution, the programmer can not do much. It does not choose/knows the sensors, neither the features nor the algorithm. Is always dependent on the API performance and ability to make the best decisions on these fields.

On the other hand, the Random Forest (RF) solution allows the programmer to do all that. However, that has a price. The implementation difficulty increases because those decisions need to be made.

The first is which sensors to use. There should be appropriated for this type of applications goal but also needs to consider the battery they spend. The solution can not only look at the accuracy, it also needs to look at the other requirements.

Related to this there is another physical limitation that requires some attention, the battery life. It is necessary to have into account which sensors to use once they spend different amounts of energy [2,17], as well as which sensor combination. An external device could be used to reduce the battery spent on the user's device once it could perform the readings and the calculations externally, sending only the results. Although, that makes the application rely on external devices and must have the communication time into consideration. If the connection fails for some reason the user does not get the result.

The third challenge comes up during the project design when the decision on which model should be chosen. Which one fits better for the solution requirements. Alongside this, it is also necessary to identify the target data, i.e., define which data is truly necessary to perform the task.

The fourth challenge is data treatment. Any machine learning system requires pre-gathered data to work. The data must be: 1) diverse enough to ensure that the model predictions capabilities accommodate a variety of possible scenarios; 2) As unbiased as possible to ensure that the model can generalize appropriately during inference, and 3) abundant. Although not all the gathered data is usable, either because the measurements were compromised due to software or hardware malfunction. So it is necessary to prepare the data, cleaning the wrong ones, select the desired features, and labeling the correct data. This improves the determination accuracy of the classification model and the app accuracy consequently.

The next challenge comes following that. The classification model needs to be trained with that treated data. However, there is no perfect solution or guide that if followed makes the solution reach 100% accuracy. The key here is in the precision mathematics (and a whole lot of trial and error) that leads to model validation through model testing.

1.3 Road Map

This thesis is is organized as follows: Chapter 1 provides an Introduction of the topic. In Chapter 2 it is discussed the Related Work. Chapter 3 presents the Architecture/Algorithm of the solution. Chapter 4 it is discussed the Implementation. Chapter 5 addresses the Evaluation methods. Finally, a Conclusion in Chapter 6.

2

Related Work

Contents

Over the past years, there has been a lot of research time and resources spent on how to recognize the current activity performed by some user. This applies to a lot of cases and has many possible solutions and uses in a users' daily life. The activity field is getting bigger since it can go from simple actions performed by humans all the way to determine the transportation mode.

Before we go any further, first we need to define/explain what is an activity and its purpose, mainly for the particular case we are addressing. An activity can be defined in two main categories, stationary or dynamic. Within the stationary category, there is, for instance, the still position, which can be standing, sitting, or lying down. In the dynamic category, we can divide it into other two large categories, the single or the combined ones (explained in more detail in Murao [18]). The single refers to activities of only one movement such as raise or wave one hand. On the other hand, combined activities are combinations of single activities performed at the same time. For instance, running or walking are combinations of moving the legs and the arms simultaneously. With the advance of technology, combined activities also began to determine the way a person moves from point A to point B. Not only just by himself but also when using a transportation mode. In other words, it can determine not only if the activity performed was a walk, but also it is possible to determine if it was driving, on a bus, or cycling. This type of activity is the most interesting for this particular case, once Biklio aims to detect cycling activities.

As an example of the activities that we will not discuss in this paper (single), they can be found in Ranasinghe [19] explaining several techniques and use cases where they are applied. Some use cases are active and assisted living and smart home systems, healthcare monitoring systems, security and surveillance systems in public areas, and sports and outdoors. All those different use cases can demand different sensory modalities to provide a suited approach, explained in Wang [20].

On the other hand, back to the desired activity type which is transportation mode detection, there are several solutions. They can be based on electromagnetic signals, such as Global System for Mobile Communications (GSM) (e.g. Sohn [21]), WIFI (e.g. Wang [22]), Bluetooth (e.g. Coroamă [23]), or a combination of them. Motion detection sensors, such as accelerometers or gyroscope (e.g. Fang [24]) can also be used to determine transportation modes. Those solutions can be external (e.g. Liu [25]) or internal (e.g. Hemminki [17]). By external, it means the use of other devices than a mobile phone, either sensors only or a combination of sensors. Internal is the opposite, relying only on sensors existing on a smartphone. Finally, over the sensors results, can be applied different algorithms. Algorithms that, based on the received sensor numbers, will determine the activity. There are many possible algorithms already implemented in this type of apps. For instance, there are several solutions based on Machine Learning (e.g. Liono [26] and Ferreira [27]) algorithms to determine the transportation mode. Some others use existing APIs like GAR [16] that can be used to accomplish the goal.

Activity recognition systems typically have three main components [28]: 1) a low-level sensing module that continuously gathers relevant information about activities using sensors; 2) a feature processing

and selection module that processes the raw sensor data into features that help discriminate between activities; and 3) a classification module that uses the features to infer what activity an individual or group of individuals is engaged in, for example, walking or cycling.

Next, we present some related work, explaining briefly how those three steps are implemented.

2.1 Sensors

Sensors reading is the first step/decision that usually needs to be made when designing an activity recognition system. It is necessary to decide which sensors to use and the number of sensors required to achieve the best accuracy. There are several types of sensors available that already were tested for activity recognition. As mentioned before, they can be based on electromagnetic signals, motion detection, internal, or external. One advantage that the electromagnetic signal sensors have compared to the motion sensors is that their sensor readings do not vary with the device placement. I.e., no matter where the device is stored while detecting an activity the values provided remain equal either if the device is in a pocket or the hand. The motion detection does not verify the same characteristic once their values are based on movement and according to the device placement can be added extra movement. The sensors gather information continuously and their output is expressed in numbers. This information means nothing to a user. For his concern are just numbers and he does not have access to it.

Usually the bigger the number of sensors used, the bigger the accuracy. However, that is not necessarily true. Either because some sensors may not be indicated for certain activity recognition or when combined with other sensors can provide information that is too similar adding nothing relevant. For instance, a light sensor may not be necessary or add much information when trying to determine if someone is walking or driving. In Saeedi [1] (Fig. 2.1) it is possible to observe the accuracy of individual/combination of sensors. Looking at the activity recognition bars (in blue) shows that more sensors do not mean higher accuracy in some cases. For instance, the accelerometer and gyroscope combination has a lower accuracy than the projected accelerometer alone. So, it is necessary to assess which sensors are essential for certain activity detection.

Each sensor spends a certain amount of battery, varying between sensors, as mentioned before. So the increase in the number of used sensors can impact the solution in other ways, especially on the battery. A solution that drains too much battery from the device where it is running is not desirable. The system should not make it impossible to use the device for other actions, for instance in the case of a phone, it should be possible to keep using it normally as if the recognition system was not running. So, it is also necessary to consider the battery consumption of each sensor to determine the best accuracy/battery combination.

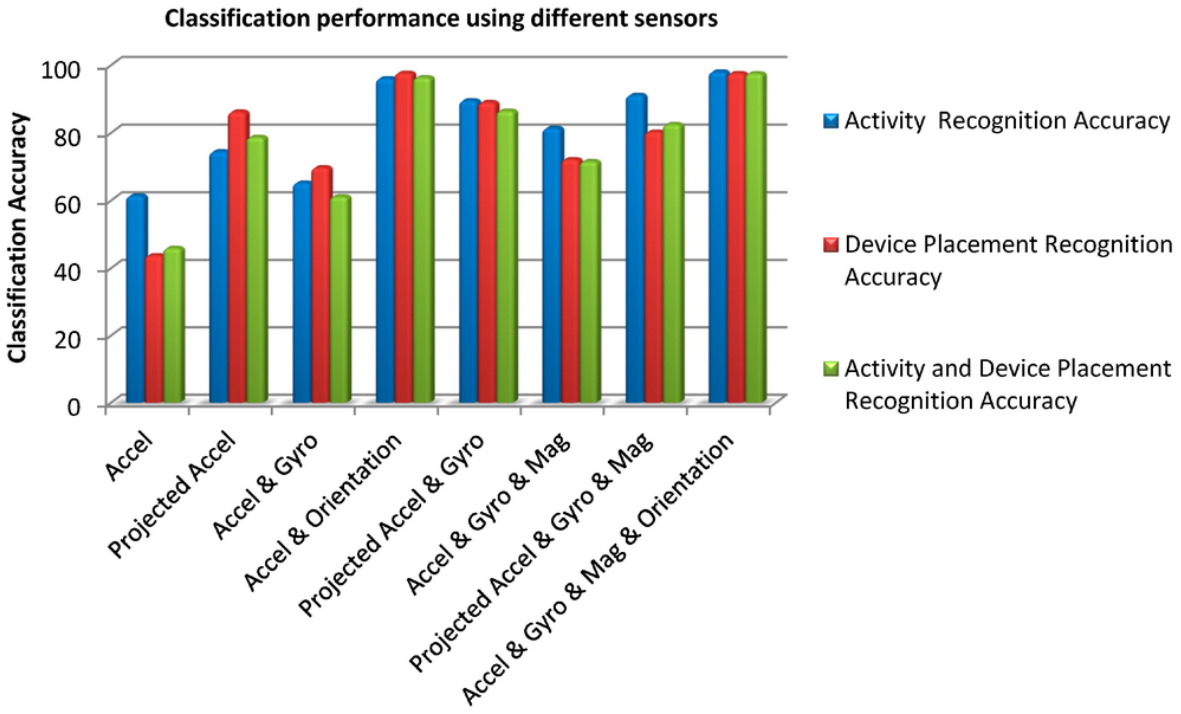

Figure 2.1: Relation between accuracy and number of sensors used (Saeedi [1]).

2.2 Features

A feature is an attribute that can be retrieved from the output of the sensors. Usually, the second step when designing an activity recognition system is to choose which features to extract. As mentioned previously, each sensor returns some numbers. The meaning varies from sensor to sensor. For example, the numbers returned from a Global Positioning System (GPS) are related to latitude and longitude but from an accelerometer are related to acceleration. Each sensor supplies many features that can be extracted, depending on how the calculations are made. For instance, two consecutive GPS readings do not give just the location differences; it is also possible to obtain the user speed by dividing the location difference by the time between both readings. Other features such as acceleration can be calculated based only on this sensor. However, not all features are necessary and more features imply more calculations, which can mean more time and battery to perform activity recognition. In Fig. 2.2 it is possible to observe that different combination of sensors leads to different execution times. So only the necessary features need to be extracted to improve the overall performance.

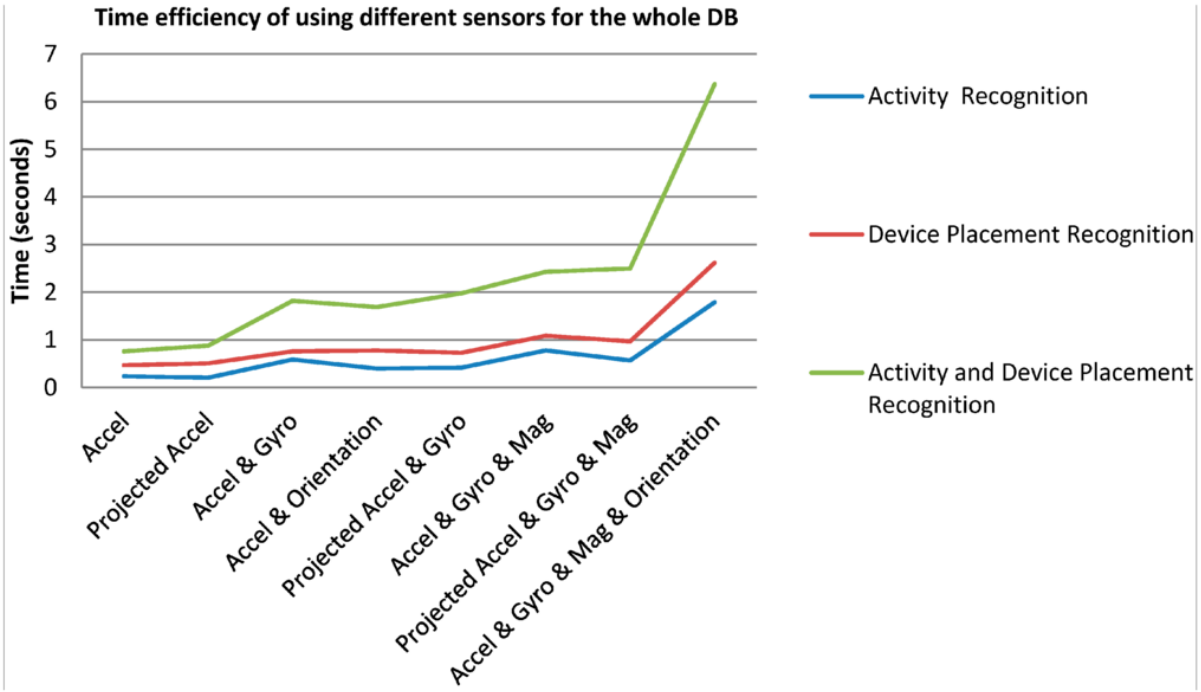

Figure 2.2: Relation between time needed to calculate the results and the number of sensors used (Saeedi [1]).

2.3 Algorithm

The algorithm choice is usually the third and final step for an activity recognition system. It is where the actual detected activity value is presented. It transforms numbers into an activity. I. e., transform the values of the received features, which does not mean anything for a regular user, into something that he understands, an activity.

There are several algorithms already used for activity recognition (e.g. Liang [29]). Such algorithms can use machine learning or not. For the machine learning-based approach there are many options, such as Decision Tree (DT) (e.g. Fang [24]) and all their variants like RF [30], AdaBoost, or Bagging [31]. Support Vector Machine (SVM) or K-Nearest Neighbors (KNN) (e.g. Fang [24]) are some examples of other machine learning algorithms applied for activity recognition. All these algorithms adapt over time to the new sensor readings since they usually save them in their storage increasing the number of samples. In other words, the machine learning algorithms are constantly learning and adapting the results based on previous results. On the other hand, there is the classical/simple algorithms [32] that do not detect patterns and need to follow some predefined rules. These algorithms are more strict and always perform the same way. I. e., the same entry value will always return the same output. Even if the entry value is a little off the defined trust interval window and it should be accepted as a certain activity, it will not. That is because these algorithms do not learn over time, unlike the machine learning ones,

so their accuracy and all their flaws will remain from the start until the end of their usage. While the ML algorithms can adapt and change their trust intervals automatically. The regular algorithms need to be changed manually by the programmer.

2.4 Use cases

As mentioned before (see Section 1.1), there are several categories for activity detection: 1) apps that encourage sustainable mobility; 2) activity tracking apps for healthcare; 3) finally, activity-driven crowd-sourcing apps.

2.4.1 Sustainable Mobility

In this section, we analyze solutions with a bigger focus on transportation modes, i.e., some of the possible ways a human can move from one place to another. These solutions can detect both motorized and non-motorized transportation modes. This makes them able to be used to encourage sustainable mobility.

Sohn [21] is an example of a solution that uses only GSM to determine a user's activity. It was tested for three different activities, `STILL`, `WALK` and `DRIVE`. This is a solution that only requires a mobile phone, concerning the user, i.e., the user does not need to carry any other device. Access Points (APs) are something that already exists and functions. The necessary features were extracted to get to one of the three previous results. Were used calculations based on the same principle as fingerprint-based location systems. More precisely, for some fixed APs the observed radio signals are consistent in time but not in space. So for a stationary return, the set of APs and signal strength should be stable. If the user is not moving the APs that his phone access does not change through time, as well as the signal strength, under normal circumstances. The exceptions can occur if an Access Point (AP) stops working or there is some big object between the phone and the AP which can cause loss/decrease of the signal between those 2 consecutive readings. On the other hand, to get a driving output it is necessary that through time the APs that are connected with the phone change. This proves movement. Changes in the signal strength prove it too. The higher the returned value, the higher the speed. A two-stage classification scheme is used. The first stage classified an instance as stationary or not. If the instance was classified as not stationary, a second classifier would determine if the instance was walking or driving. This was to present better results compared to the one-stage approach. DT [24] was used to classify those instances. To improve the recognition rate was added one boosting algorithm named AdaBoost [31]. It is a more accurate selector because it is a sequential process, i.e., trees are grown using the information from a previously grown tree one after the other. This process slowly learns from data and tries to improve its prediction in subsequent iterations. It combines other algorithms ('weak

learners'), DT in this case, output into a sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are adjusted in favor of those instances misclassified by previous classifiers giving them more weight. Although it is sensitive to noisy data and outliers, i.e., corrupted or distorted data which can be a problem in case a sensor is damaged.

Anderson [33] takes a similar approach using only GSM data to detect the travel mode and basing its output on the same two variables as the previous example, the nearby APs, and the signal strength variations. Affirming that higher variations, either AP cells or signal strength, equal to a higher speed. It can detect the same three activities above. Each activity has different signal strength fluctuations, higher for driving, and lower for still. With the use of previous data, it can increase reliability because it is possible to filter drops on the signal strength when they occur, helping to identify the driving activity over the walking activity when big drops are detected. This may be associated with the stop-start nature of driving in busy urban environments where the flow of traffic is controlled and roads are congested. Where it starts to be different is in the third step of an activity recognition system, the algorithm. In this case, the method chosen to predict the current activity is different and works as a classifier. It is used the Hidden Markov Model (HMM) [33] for that purpose. The HMM is a statistical Markov model in which the system being modeled is assumed to be a Markov process (X) with unobservable ("hidden") states. HMM assumes that there is another process Y whose behavior "depends" on X. The goal is to learn about X by observing Y.

Coroamă [23] aims to study the usefulness of Bluetooth (BT) and WIFI signals to calculate the transportation mode. It can detect six activities, `TRAM`, `BUS`, `WALK`, `TRAIN`, `CAR`, and `BICYCLE`. Here proximity patterns (e.g., the share of WIFIs that stay in range for longer periods or their rate of change) are the interesting features, assuming that this share is greater for specific transportation modes such as public transportations once there are more people and consequently more devices. The implementation consists simply of repeatedly sending signals, BT and WIFI, to determine the users' activity based on the signal strength obtained and variation of connected devices. The extracted features were based on BT and WIFI, making some assumptions. On the WIFI assumes that constant APs change means movement and the higher the number of different APs through time the higher the speed. BT assumes that public transportation modes have higher BT devices and they tend to stay together for a certain amount of time. For example, on a train, the users can not move away from the route causing the devices to stay together. On the other hand, if the user is walking the probability of some other five persons move alongside him for a certain amount of time is lower. Weak overall performance from this leads to considering other solutions. One is based on GPS and an accelerometer. The other was the combination of both, BT , WIFI, GPS, and accelerometer. For all the solutions was used the same classifier, the RF. In most of the transportation modes, the desired solution had poor performance compared to the existing one based on GPS and Accelerometer. Except for the train, where it exceeded the existing solution. That

was due to the relatively stable collection of BT devices close by compared to the not so stable GPS signal. However, the best overall solution was the combination of all the four sensors, which ended up by support the initial intention of using BT and WIFI to improve the transportation mode classification.

Montoya [34] work can detect FOOT, CAR, BIKE, BUS, TRAIN, and TRAM activities. A big part of its decision algorithm is based on BT and WIFI signals, especially BT by sending signals repeatedly to infer signal levels. It is assumed that BT devices are carried by almost every person, such as phones, headphones, or smartwatches. In some cases, a user can have more than one. It assumes that this signal continuity from the same devices is bigger in some transportation modes (mostly public) because normally when walking we tend to move alone but in public transportation, we move in groups staying connected to the same devices more time. The values returned from BT and WIFI helped to differentiate between public and non-public transportation. Though this is not enough so other sensor data were included in the calculations apart from BT and WIFI such as GPS, accelerometer, and cellular alongside the public transportation routes. The final decision is also significantly based on the GPS sensor. This happens since initially, it is gathered information relative to the public transportation systems, more concretely their routes. These routes are available to public knowledge and anyone can have access to them. This increases public transportation accuracy. For instance, if consecutive coordinates are following a specific known Bus route, it increases the Bus probability. Based on those arguments it is calculated the current activity. That is performed in two steps. First, filtering using a dynamic Bayesian network (DBN) [35] that models the probabilistic relationship between paths in the Transportation network and sensor data. The result is a path in the Transportation network employing the following modes: foot, bicycle, road_vehicle, and rail. Second, smoothing this path, refining it by matching segments recognized as road_vehicle or rail with public transportation routes of type bus, train, metro, or tram obtained from the public transportation routes. The highest probability option is returned as the current activity.

A different approach was taken in Stenneth [36] where they have used GIS information models alongside the GPS information. The information provided by GIS is transportation network data, represented as real-time locations of buses, rail line routes, and bus stop locations. This improves the distinction between public and non-public transportations. The detected activities are WALK, BUS, DRIVING, TRAIN, STATIONARY, and BIKE. If the detected activity is BUS they even go a bit deeper into the classification, providing also the actual bus the user is traveling at the moment. The transportation mode determination is divided into two stages, the learning and inference stages. In the first stage, the GPS data reported by the user's device is merged with the transportation network data and respectively labeled ground truth. This data is used to train the classification model. The GPS data is collected every t seconds, registering it as a new sample. In the second stage, the system determines the transportation mode by getting the merged data and analyzing it sample by sample. It was used a machine learning toolset to evaluate the different classification models tested on transportation mode detection. The results indicated that

the RF [30] was the best choice reaching a higher accuracy value. Although this system may not provide real-time information to the user once the transportation mode inference is performed in a server (remotely) and not on the phone (locally).

Another example of the GPS field to determine transportation mode is Liang [29] that can determine up to six different modes. `WALK`, `BIKE`, `BUS`, `CAR`, `TRAIN`, and `PLANE` respectively. This example, as the previous one, also uses GIS to get real-time information about public transportations, more precisely the buses. The information available can be such as bus stops or bus routes. This facilitates the identification of the transportation mode, being able to identify precisely the bus activity. The information gathered from the GPS is labeled with the respective timestamp and its coordinates are compared with the known coordinates from the public transportations to see if there is any resemblance. For that, it was implemented a trip segmentation method including trip identification and transition point detection. This consists of splitting the entire trip into smaller ones of only one transportation mode. The points where the trip is split are transition points, where a user changes transportation mode. After that, on each segment, several features are selected and run through the classification model. In this case, the choice was the Random Forest (RF) [30] over others like SVM [24], AdaBoost, or Bagging [31]. Finally, it is considered the adjacent segment (preceding and following) to correct possible errors on the classification model output to improve the system accuracy. The flow is the following: first, the output of the RF is passed through the correction processing; second, the correction processing and the RF outputs are merged; finally returns the final activity result. This correction step considering the adjacent segment helps to eliminate some outliers that appear in the middle of an activity. For instance, a user is detected walking for several classifier outputs and in the middle for one or two classifier outputs the result is driving. That is not physically possible and that value is correctly eliminated. However, this implies that the response to the current activity can not be immediate.

Zhou [37] can determine up to six different travel modes based on GPS. Those are `WALK`, `BICYCLE`, `E-BICYCLE`, `CAR`, `BUS`, and `SUBWAY`. Firstly, GPS raw data is collected from the user phone, containing timestamp, latitude, and longitude. Any incorrect measurements and travel practices were deleted from the storage, for instance, if it is missing some information in one of the collected fields. After cleaning the data, for every tracking point are extracted features, such as coordinates, calculated and compared with previous and following (adjacent) tracking points to assess whether the features are similar. Those points that present large differences are run through a machine learning model, RF [30] to be precise, to identify them as transition points between different travel modes. This trains the classifier to identify transition points. After that, it is possible to split into single-mode segments. For each single-mode segment are calculated their features to retrain the machine learning model and recognize each travel mode. Only three-quarters of the samples are used for model training. The other quarter is used for testing the model.

A different approach was taken in Hemminki [17] compared to the previous example. It was used the existing accelerometer on the user's smartphone instead of an external one. It detects seven different activities. `STATIONARY`, `WALK`, `CAR`, `BUS`, `TRAIN`, `METRO`, and `TRAM` respectively. To determine the performed activity they first use a kinematic motion classifier that performs a coarse-grained distinction between pedestrian and other modalities. This simplifies the complexity of the problem. Like other works are using more than one step to perform the activity inference. If there is no substantial physical movement, the process goes to a stationary classifier, which determines whether the user is stationary or in motorized transport. If the motorized is detected it goes to the motorized classifier which is responsible to determine the motorized vehicle. It is assumed that changes in transportation behavior typically occur infrequently and each activity has a duration of several minutes. Furthermore, changes from one motorized transportation modality to another are typically separated by easily detectable walking segments. The route is split into segments, each representing a travel mode. For each segment are calculated frame-based, peak-based, and segment-based features. Frame-based can effectively capture the characteristics of high-frequency motion. Peak-based can identify lower frequencies, such as acceleration and breaking periods of motorized vehicles. Segment-based characterize patterns of acceleration and deceleration periods over the observed segment. For the kinematic motion classifier (the first of three classifiers) is used the discrete Hidden Markov Model (DHMM) [38]. The other two, stationary and motorized classifiers use segment-based classification where a simple voting scheme is used to aggregate frame-based classifications over the observed segment. The frame-based classifications are obtained using an instance-based classifier. Each of the three classifiers considers a variant of AdaBoost as the instance-based classifier that outputs a numeric value indicating the voting result of the weak classifiers.

Another example is Shin [39]. This one uses the mobile phone sensors to calculate the transportation mode. This specific case does not use GPS to save energy and maximize battery usage time when determining the travel mode. Instead, a network-based location system is used. It can be used both indoor and outdoor for most locations and weather conditions alongside an accelerometer. In other words, it uses the mobile phone network provider's infrastructure to determine its location. To locate a phone using that technology it is measured the signal strength and antenna patterns. Also, it uses the concept that the phone always connects with the closest tower. So knowing the tower location implies knowing that the phone is not far from there. As some of the previous works discussed utilizes the segmentation technique. Splits the sample into smaller ones by identifying walking periods, called the transition period. This is performed based that the mean acceleration while walking is typically much higher than on any other activity. Then splits the rest of the trace with those walking periods to obtain the other activity segments. Next analyze each one individually, extracting mainly speed and acceleration features and calculating the most probable activity. Based on average speed it determines if it is either a vehicle riding activity or not. In case it is considered a vehicle riding activity, it is calculated the probability

of each type tested, either train, car, or bus, based on the acceleration. It is estimated an acceleration profile for each mode and used to classify particular acceleration behavior into one of the recognized travel modes. These calculations are performed on a server, detecting the following activities WALK, TRAIN/TRAM, CAR, and BUS.

Fang [40] uses the accelerometer, magnetometer, and gyroscope sensors embedded in a smartphone. It measured five distinct transportation modes including STILL, WALK, RUN, BIKE, and VEHICLE. This last category is divided into smaller categories, MOTORCYCLE, CAR, BUS, METRO, and HSR (High Speed Rail) respectively. The transportation mode classification system was split into two stages, offline and online. The offline stage consists of extracting the desired features from the sensors and label them, training the Deep Neural Network (DNN) model [41]. The DNN is a neural network with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship. The network moves through the layers calculating the probability of each output. This model is what will classify future activities based on known previous activities. In the online stage, the real-time sensor values are turned into the desired features and run through the DNN model to calculate the most probable transportation mode. Five different algorithms were tested to measure their accuracy. The DNN outperformed them all, showing better results in terms of accuracy and acceptable model size.

Fang [24], just like Fang [40] uses the accelerometer, magnetometer, and gyroscope embedded on a smartphone to determine the travel mode. This work can determine up to 10 different modes of transportation, STILL, WALKING, RUNNING, BIKING, MOTORCYCLE, CAR, BUS, METRO, TRAIN, and HSR. Yet for test classification, only five modes are considered, gathering MOTORCYCLE, CAR, BUS, METRO, TRAIN, and HSR as a single-mode called VEHICLE. To train the classification model it was used three machine learning algorithms, including DT, KNN, and SVM [24]. For each algorithm was tested two different sets of features and conclude that from all, the sets with the biggest amount of features had better results. The algorithm that presented better accuracy results for transportation modes prediction in both sets of features was the SVM. However, was much slower and memory heavier to predict those results than DT. But to distinguish between the VEHICLE category KNN outperforms SVM with a shorter prediction time, but a bigger memory size.

Yu [2] is another solution that uses machine learning to get travel modes. This particular work is very focused on minimal computational complexity, memory, and battery usage. This is really important once it handles some of the challenges mentioned in the challenges section, such as battery life. As Fang [24] detect 10 different modes of travel, STILL, WALKING, RUNNING, BIKING, MOTORCYCLE, CAR, BUS, METRO, TRAIN, and HSR. But also combined the last 6 modes into one, the VEHICLE. It uses three different sensors, an accelerometer, a magnetometer, and a virtual gyroscope. This last one is a combination of the first two sensors as a way to decrease battery consumption once it consumes less than a regular

gyroscope. From each sensor were extracted features that were tested for three different algorithms, DT, SVM [24], and AdaBoost [31]. DT is a piece-wise linear model, AdaBoost smooths the decision boundary of DTs, and SVMs can work with a kernel to adjust its model complexity. The SVM presented the best result among the others. It is performed an error correction over the classifier results. This is due to the possible misclassification in moments that are too close to a change of activity moment. In other words, when an activity is changed from one mode to another, the moving window that straddles the two modes during the transition can include features from both modes. That correction is performed by voting on each activity type. In the beginning, all of the scores are set as zero. At each time point, the score of the type predicted by the classifier is increased by one unit. In contrast, the scores of the other types are decreased by one unit. The maximum voting values each type can reach is four and the minimum is zero. Then, the type with the highest score is determined as the modified prediction.

2.4.2 Activity Tracking

Many solutions try to detect human movements, i.e., movements the human body can perform. Some are more focused on physical activities, such as walking or run, others on daily movements such as climbing escalators. They can be used to track the users' daily activities for health reasons. Some of the detected activities here also can be used as a means of transportation, so it is important to see some solutions towards this type of app.

Murao [18] is one of the cases that use only accelerometers to detect activities. In this particular case, it was tested for various movements. The most relevant are: `SITTING`, `STANDING`, `LAYING`, `KNEELING`, `WALKING`, `RUNNING`, `CYCLING`, `DESCENDING` and `ASCENDING STAIRS`. The rest was classified as single activities at the beginning of Section 2. In this document, such activities are said to be local. The solution proposed by Murao consists of attaching five accelerometers to a user. One on each wrist and ankle, and another one on the hip. The system has three phases. The first phase classifies the activities of each part of the body into three types: posture, behavior, and gesture. Posture is a state of a user remaining stationary lasting for a certain length of time, e.g. sitting. Behavior is a state of a user doing periodical movement lasting for a certain length of time, e.g. walking. A gesture is not a state but once-off action having a starting point and endpoint that sporadically occur, e.g. punch. The SVM [24] classifier is applied to posture and behavior. The second phase recognizes activities, in each sensor, according to the activity type. That is performed by having a mean sample value, maximum, and minimum value that delimitate a posture called an epsilon tube. If the sensor readings fit outside the tube it means that it is a gesture or behavior. To distinguish between those two is necessary to observe the constancy. The gestures present higher and more picks of acceleration because they are punctual movements where the behaviors are not, they are performed for a significant amount of time. After each sensor/body part has its activity detected is necessary to start the third phase, combining the recognition

results and outputs the overall activity. This is done by attributing one vote to each body part. Each vote is weighted based on the activity recall and the same activity values are summed. The activity with a higher score is the overall activity.

Liu [25] uses two accelerometers, one UV sensor, and two piezoelectric sensors to determine users' activity. All these sensors are external. This solution was oriented to detect physical activities, but we can consider some of them like `WALK`, `CYCLING` or `RUN`, given that they fit into our activity definition. The UV sensor measures the environment's light conditions. The piezoelectric sensor measures the expansion and contraction associated with ventilation (breathing rate and volume). Helps the system to detect if a user is struggling when performing such activities. The data provided by the sensors is saved in one SD card for later calculation and analysis. Each sensor has its sampling rate, i.e., the time between readings. This is an important parameter because it can affect the system. For instance, a high sampling rate contributes to good signal fidelity under the physical activities of various intensities. It also leads to requirements for higher data storage capacity, higher energy consumption, and hence shorter battery life. So the correct sampling rate must be used. SVM was the chosen classification model to determine the activity with a 30 seconds sliding window. In other words, the sensor readings are delayed 30 seconds. To train the classifier was used a "leave-one-out" protocol. It consists of training using all data points except one, and then test with the left-out data point and was repeated for all the subjects.

Choudhury [28] (just like the previous example) uses an external device to accomplish the goal of detecting activities. Two different versions of the device were used. The final version was composed of the Electret microphone, Visible light phototransistor, 3-axis digital accelerometer, Digital barometer temperature, Digital IR and visible+IR light, Digital humidity/temperature, and Digital Compass, 3D magnetometers, 3D gyros, and 3D compass. It was used to detect daily activities, but the most interesting ones for our case are `WALKING`, `RUNNING`, and `CYCLING`. Some features were selected from them and run through the boosting algorithm [31]. Boosting is a general method for improving the performance of any learning algorithm. In theory, Boosting can be used to significantly reduce the error of any "weak" learning algorithm that consistently generates classifiers which need only be a little bit better than random guessing. It works by asking several weak learners the answer to some questions and combining those answers into one. Weak learners are classifiers slightly correlated with the true classification. The trained system outputs the probability scores for different activities that might be occurring at a given time, returning the most probable activity. The final inference version of this work does not need that every sample is labeled with corresponding ground truth. This is a good advantage because usually, the labels are placed by humans. This implies a significant reduction of time to prepare the samples for training and consequently the training phase duration.

Reddy [38] is another work in transportation mode detection category. There are detected several activities such as `STILL`, `WALKING`, `RUNNING`, `BIKING`, and `VEHICLE`. It is mainly focused on this cate-

gory except for VEHICLE that fits into the Transportation Mode category. It uses information from an accelerometer and a GPS to achieve that. It uses a third sensor, GSM, only to detect outdoor activities. By detecting changes in the cell towers it detects that the user is outside traveling. This solution has a classification model that is trained and later determine activities using the sensor readings obtained from the accelerometer and the GPS. When the user is detected outside, the system automatically turns on the transportation mode classifier. This option can save battery once when the user is indoors the only sensor that is working is the GSM. To select the classifier to be used, three different types of classifiers were compared. Instance classifiers such as DT, SVM, continuous Hidden Markov Model (CHMM), and a two-stage system involving the most accurate instance-based classifier (which is the DT after some tests) combined with DHMM. After testing all these algorithms the DT+DHMM was the one that presented better results with 93.7% and 93.6% in precision and recall respectively. Not very far were the results obtained from the DT only with 91.3% for both precision and recall. To recap, the system overall structure of the classifier is the following: 1) reads from the sensors; 2) filters the noise on the sensor readings; 3) calculates the features of those cleaned signals; 4) run them through the DT classifier; 5) run the DT classifier output through the DHMM classifier; 6) and return the activity.

Another existing work on the machine learning field to detect transportation modes, more focused on a healthy lifestyle, is Parkka [42]. It can detect several activities related to human daily life. We are going to focus on those that can fit our activity definition, such as WALK, RUN, and BIKE. It is an external implementation, i.e. it is based on wearable sensors external to the user's phone. It used many sensors in many body parts. For instance, it uses an IR light absorption in the finger and an IR light reflectance in the forehead to measure heart rate, or a GPS on the shoulder to know the user's location. A large number of samples were gathered and labeled by an annotator that followed the testers through their tests and wrote changes in context. This approach helped the writers to familiarize themselves with each sensor's readings because they could be compared with the context notes. It also helps to choose the best features according to your intentions. The sensor readings were labeled with the respective context. For activity classification were tested three different methods, automatically generated DT, custom DT, and Artificial neural network (ANN). The custom DT was built after analyzing and labeling the samples. After the classification is performed a post-processing step that eliminates all short activities because it is assumed that activities that only last for a few seconds are not realistic and they would cause misclassification. The automatically generated DT performed better achieving 86% overall accuracy. It was also concluded, by testing several different sensors that the most information-rich and accurate sensors for activity recognition are accelerometers.

The GAR API [16] is a low-level application that reads the outputs of the physical sensors existing on a smartphone. This app reads all types of sensors and calculates the performed activity at the moment. The API can read from a single sensor or multiple. This depends on the hardware available on each

smartphone. When it has into consideration more than one sensor, it combines the results of each one. Those results can calculate a more accurate and precise activity. The API uses machine learning algorithms to calculate the results. This way can adapt to different outputs over time. It also improves its accuracy along with the number of times used. To optimize resources, if no movement is detected for a while, the API may stop activity reporting. To start reporting again uses low energy sensors when it detects movement.

There are two types of GAR APIs, the transition and the sampling. They are very similar but have some differences. The Transition is an easier model once it requires less work to implement. The programmer only needs to call the API and it will do the hard work and decisions. It improves accuracy, consumes less power, and enhances engineering productivity. Although that implies that the programmer is stuck to the API, it can not change anything.

The Sampling API is a little bit more flexible. It allows the programmer to access unfiltered/raw data and also to control the frequency at which the activity recognition is running. Although on this API apps are responsible for managing power consumption by defining the detection interval and writing a filter on top of the raw activity classifications as individual predictions may be noisy. The transition API handles these for you.

After receiving the values of the sensors, the machine learning algorithm is applied over them calculating the activity probability. Eight different types of activity could be detected by the algorithm. `IN_VEHICLE`, `ON_FOOT`, `RUNNING`, `WALKING`, `ON_BICYCLE`, `STILL`, `TILTING` and `UNKNOWN`. For each of these, a probability is calculated. That corresponds to the probability of each one of those activities are being performed at the moment. The values for each type will be between 0 and 100 in percentage. The higher the value, the certain the system is about the performed activity. However, some activity types are not mutually exclusive, for example, a user can be walking while on a bus. Another concern is that some of them are hierarchical, such as `WALKING` and `RUNNING` are concrete cases of `ON_FOOT`. This could lead to multiple activity types to return a high probability value. Which makes it harder to define the exact activity at the moment. The minimum acceptable value to which the activity type is considered correct is defined by the programmer. These calculations are performed several times within a time interval defined also by the programmer or not. This step depends on whether it is being used the transition or the sampling API. This time has a direct impact on smartphone performance and also on its battery duration. Usually smaller intervals mean more battery consumption but also better accuracy. Greater intervals lead to the exact opposite.

As mentioned before, to preserve battery the readings stop when the device is `STILL` for an extended period. In other words, the sensor values do not change for some time because the sensors stop working for the API purpose. Once the device moves again it will resume. Though, this only happens on some smartphones that own a specific type of sensors, the `Sensor.TYPE_SIGNIFICANT_MOTION` hardware.

2.4.3 Crowdsourcing apps

The high number of people owning a phone with sensors and using the same app allows them to share relevant information between users. In other words, it is possible to share information of one user with others, relative to traffic or parking spots between users of the same app. This type of app, for example, helps the users to manage their routes based on current traffic, enabling them to lose less time stuck in traffic.

One example of the use of a crowdsourcing app, for transportation mode detection, is Zheng [43]. It was tested with four different activities: `WALK`, `BIKE`, `BUS`, and `DRIVING`. It consists of getting the trace performed by the user based only on the GPS available on his phone. After the complete trace knowledge, the system starts dividing it into two categories, walk, and non-walk. At the first stage, the authors do not differentiate between bike, bus, and drive to reduce the segmentation complexity. That is because walking is seen as a transition activity, i.e. it is used to change between one of the other three activities. The point that separates the walk from the non-walk segment is called a change point. It stands for a place where a user changes their transportation mode in a trajectory. After detecting these segments and the change points, the non-walk segments are analyzed to determine, in real-time, the specific transportation mode. For that, some features are calculated on each segment and run through a classifier. From this point are tested two alternatives. The first is the use of general classifiers, like DT, SVM, and Bayesian network (BN) to infer considering segments of GPS tracks as independent instances. After the inference, post-processing, which takes the transition probability between different transportation modes into account, is implemented to improve the prediction accuracy. The second option is GPS data are considered as a kind of sequential data. Conditional random field (CRF), a framework for building probabilistic models to segment and label sequence data, is leveraged to perform the inference. Since the conditional probabilities between different transportation modes have been considered in the CRF graphical model, the post-processing is not performed. Although the DT presented better results in classifying transportation modes. All these calculations are performed on the server.

2.4.4 Discussion

Table 2.1 provides a brief overview of all the mentioned related work. It is focused on cycling activity and the obtained values. It provides information about the used sensors, if machine learning algorithms were used, and if it were which algorithm was. Finally, it provides the obtained accuracy alongside if it is local or not. In some cases, it was not possible to know the accuracy once the values are not mentioned in the work references. The same happened in some cases for the locality variable. Those are represented as NS (not specified).

After analyzing these works it is necessary to understand the advantages and disadvantages of each

Cycling				
Reference	Used sensors	Accuracy	Local	Use Machine Learning
[34]	BT and WIFI	98%	YES	DBN
[23]	BT, WIFI, GPS, Accelerometer	96.6%	YES	RF
[23]	BT, WIFI	82%	YES	RF
[23]	GPS, Accelerometer	90%	YES	RF
[43]	GPS	75.4%	YES	DT
[36]	GPS	93%	NO	RF
[29]	GPS	90%	NS	RF
[37]	GPS	90%	NS	RF
[18]	Accelerometer	82%	YES	SVM
[25]	Accelerometer, UV, respiration	NS	NO	SVM
[28]	various	NS	YES	NO
[40]	Accelerometer, Magnetometer, Gyroscope	88.6%	YES	DNN
[38]	GPS, Accelerometer, GSM	92.8%	YES	DT+DHMM
[24]	Accelerometer, Magnetometer, Gyroscope	94.09%	NS	KNN
[42]	various	82%	NS	DT
[2]	Accelerometer, Magnetometer, Virtual Gyroscope	85.39%	YES	SVM
[16]	various	NS	YES	NO

Table 2.1: Used sensors and Machine Learning algorithms to determine cycling mode with corresponding precision accuracy

solution. See if they are a deal breaker to our goals or not. For instance, if some sensor is not suitable or does not work to detect the cycling activity. That allows choosing the most suitable solution for this work.

All these works have something in common with Biklio. They all detect transportation modes using different sensors and algorithms. After analyzing them we can understand that are many different techniques that can be used on Biklio. However, there are as well some that are not as suitable to Biklio, due to its characteristics. For instance, Biklio's main focus is on cycling activity. All other activities do not matter for this work.

It is important to know the strengths and weaknesses of each type of sensor as well. For instance, the GPS sensors present a good location precision. It can help to track the route performed by the user or even help it to navigate to his destination if that is the case. But it has some problems in big cities. Cities with high skylines can be problematic because a building may be covering the GPS satellite. This can cause the signal to not be received, causing a loss of connection for a certain amount of time. The GSM, in contrast to GPS, has its coverage available throughout most, if not all, of a person's day and does not require a line of sight to work. It is also less power-demanding than GPS which will end up saving battery. However, it is not as precise as some other technologies, such as GPS. Works better in

populated areas due to the bigger crowding of antennas/cell towers. And finally but most importantly, it can be affected by the weather. On rainy days the accuracy may drop. This almost restrains the app only to sunny days if intending to keep the detection accuracy as high as possible. The WIFI presents a good accuracy indoor. It is easy to create a WIFI network. Compared to GSM requires lower money and time expenses, for system assembly. Although it does not work so well outdoor, which is very important for Biklio once it aims to detect outdoor cycling activities from point A to B. It is really hard to apply outdoors due to the inexist/small wifi network, particularly in underpopulated/less dense population areas. Objects between the receiver and the sender can affect the signal.

The same applies to the algorithms used, where the Random Forest presented some encouraging results. Also, the algorithm requires little prior knowledge, is insensitive to noise, abnormal values, and correlated features. What this means is that if for some reason one of the sensors does not respond in time and the value to calculate a current activity is null or zero the RF can still calculate it without losing much of its accuracy. This algorithm needs the programmer to choose which sensors and features must be selected. It provides greater freedom for the programmer to make his choices. However, that also means that the programmer is responsible for battery consumption and performance.

Another interesting solution is the GAR. This one because it reduces the programmer's work by managing the sensors and features selection as well as the detection algorithm. It also has some pre-built mechanisms to save battery, which helps the programmer to reduce the app energy consumption. Although GAR presents some drawbacks. First of all, it is a type of black-box algorithm. In other words, the GAR does not let the programmers know what is happening. They know that they are reading from sensors but do not know from each sensor. They know that the activities are being calculated based on features and calculations but do not know which features and calculations are being performed. All these factors make the programmer work harder when it comes to improving the algorithm.

Both algorithms/solutions seem pretty interesting for cycling detection. It is important to keep in mind that we must fulfill this work requirement. It is necessary to obtain the highest accuracy possible and both the GAR and RF present some interesting values to that matter. This would fulfill the first requirement. The second requirement implies low response time and that can be obtained by running all locally being independent of other devices execution times and communication between devices. Both solutions can be applied over a local implementation that meets the second requirement. Both solutions, RF more than GAR, give the programmer room to make choices that can impact battery usage. So once again both meet the third requirement, which aims to the lowest battery consumption possible. Finally, for the fourth and last requirement that the app should be run on an Android device, both solutions meet that. They can be implemented for that target Operating System (OS). So it is possible to observe that these solutions can meet all the requirements, giving them the chance to be considered as possible candidates for this work.

3

Architecture/Algorithm

Contents

In this chapter we provide details respecting the architecture and the algorithm of the solutions, once there were implemented two different solutions. Both solutions aim to determine if the user is cycling or not. The first uses the GAR API. The second uses a ML algorithm called RF. Their differences are explained in this chapter when it comes to architecture and algorithm and is explained in Chapter 4 for the implementation differences. They were later compared and evaluated in Chapter 5 to determine which one was the best solution.

Firstly, the design requirements are discussed and how they can restrict/limit the design and implementation of the application. It is also provided a brief background to Biklio's current application explaining its most relevant parts. Then, it is explained how the app is structured as well as how it works, namely how the solution behaves according to different scenarios. In other words, the architecture and algorithm structure and behavior. It is important to note that the architecture and algorithm is explained for both GAR and RF solutions. This is since both solutions were implemented and both system's details must be shown to understand their main differences. It may help to perform a comparison between them and understand them better for this type of application, which aims to detect cycling activities. We should also clarify that for architecture and algorithm analysis we are only considering the GAR without the algorithm described next in the Section 3.2 and the RF solution. This way it is possible to provide a direct comparison between them, for later comparison between their results in several fields in the Chapter 5.

3.1 Design Requirements

Before thinking of a specif algorithm and designing its solution, first, it is necessary to assess if the desired solution/algorithm fulfills all the criteria. As specified in Section 1.2 some requirements need to be satisfied. Those requirements are namely: 1) high accuracy; 2) low response time/delay; 3) battery; and 4) software compatibility. The specified requirements must be taken into consideration because they may affect or limit the design of the application. To understand that better we may need to take a closer look at the requirements.

For instance, if we look at the first requirement, it states that the solution must present or look forward to an algorithm that can meet it for cycling detection. The algorithms must be evaluated to understand if they are aimed for that purpose. This is because an algorithm can be better for a specific type of work than for others. The second requirement may have a higher impact on the architecture design of the solution. That is due to the restriction of time delay. Maybe an independent solution, i.e., that performs everything locally without any external communication may be the most suited for this case. This way by performing all the computing locally the solution is not vulnerable/dependent on external factors, such as bad connection, bad weather which can also impact some communications, between

many others. The third requirement asks for a battery-aware solution. In other words, a solution that must take into consideration the average user's daily usage of the phone for other tasks such as making calls or texting. This is a bit contradicting the previous, once the best way to save battery would be doing everything externally and only send the result in the end. This way the calculations would not be performed in the device which could improve battery consumption. Although that would increase the number of communications and dependency of the solution to external devices. Here a decision must be taken, which requirement weights more for this work's purpose? Once the main goal for this work was always to use only the mobile phone because it is more practical and inexpensive for the average user. If an external device was necessary the user would have to buy that extra device and install the app instead of just installing the app. So for that reason, it was given more importance to the second requirement. Though, this does not mean the third solution must be ignored or forgotten. It means that the solution must be adjusted to using only the mobile phone computational power the minimum battery must be used. That can imply reducing the number of samples, i.e., increase the sampling interval once each reading consumes battery. Also, different sensors spend different amounts of battery, so the sensors used are also important here, as shown in Fig. 3.1. There is possible to see the energy spent by each sensor, namely the Central Processing Unit (CPU), Microcontroller Unit (MCU), GPS, WIFI, gyroscope, magnetometer, and accelerometer. The MCU contains one or more CPUs (processor cores) along with memory and programmable input/output peripherals. The RF solution must consider this information and chose the sensors according to the requirements. The GAR solution does not need this because the sensors are pre-defined.

Based on this information, the sensors choose for this work were the accelerometer and the GPS once they have proved, in other works, to present high accuracy. The accelerometer is a very common sensor for this type of works because it is low battery demanding and presents good results. The GPS is also another common sensor used for this purpose because it is very precise and provides necessary information although this one is not so battery friendly. GPS is a bit power-demanding compared with the other sensors. But once it provides essential info it is still used but not as frequently as the accelerometer to preserve the device battery at the maximum without compromising too much the solution accuracy. The fourth and final requirement just states that the solution must be implemented in Android, so the necessary android libraries and functions must be used. Android was chosen because of its programming languages. It has Java has one of its official languages and Java is a very well known programming language as well it owns several libraries that allow the implementation of both solutions. That is explained in more detail in Section 4.1.

	Power	Condition
CPU (running at 1.4GHz)	88.0mA	Active status
	5.2mA	Idle status
MCU (running at 16MHz)	0.5mA	Active status
	0.1mA	Idle status
GPS	30.0mA	Tracking satellite
WiFi	10.5mA	Scanning every 10 sec
Gyroscope	6.0mA	Sampling at 30Hz
Magnetometer	0.4mA	Sampling at 30Hz
Accelerometer	0.1mA	Sampling at 30Hz

Figure 3.1: Sensors power consumption (Yu [2])

3.2 Background

As mentioned previously, Bikilo is the app of interest once it aims to detect cycling activities, and is used to test our solution. So firstly the app must be introduced to explain what it is and how it works. This provides some basic notions about the app's behavior. Next, a different strategy on how to improve its overall accuracy is discussed.

The current algorithm has five states, the Eligible, the Ineligible, the Vehicular, the Cycling, and the Eligible Cycling State (Fig. 3.2). The Eligible state is a transition one where the user was detected moving, either cycling or in a vehicle, but currently, it is not any more or it is not a continuous movement. The Ineligible state is the one where the user is not detected moving. I.E., when his movement has stopped for a long time. The Vehicular state is the one where the users that are on a vehicle are placed/designated. The Cycling state is the one designated to the users cycling for a small amount of time and has not earned the right to be in the Eligible Cycling one. The Eligible Cycling state is the one that confirms the user has earned the right to claim some kind of discount or gift. This state only applies to users detected cycling for a while.

The transitions between states are the following. Starting from the Eligible State, it can go to Ineligible, Vehicular, or Eligible Cycling State. Although it cannot go directly to Cycling State, it needs to go to Ineligible first. Ineligible can only maintain or go to Cycling. Cycling can either go to Eligible Cycling or Ineligible. Eligible Cycling can only maintain or go to Eligible. Finally, Vehicular can go to Eligible or Ineligible. This algorithm is applied over the GAR API [16].

Concerning architecture, i.e., how the app is organized and where each operation is performed, Biklio has a basic client-server communication but operates mainly locally/on the client. In other words, the app performs the current activity calculations on the device where it is running. Both the GAR and the previously explained algorithm are executed in the device. With this, the app tries to minimize the number of connections established with the server aiming to reduce the response time. This way it does not depend on communication conditions but only on the device's capability to calculate the current activity. The server only performs critical steps such as login, and upload/download travels info.

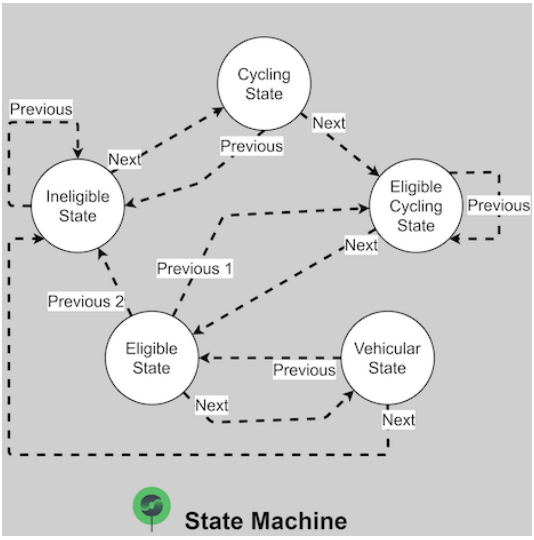

Figure 3.2: Current Biklio Transitions algorithm

This type of architecture allows the user to know the recognized activity in real-time, depending on the power and capability of the device to do such calculations as fast as possible. However, this type of implementation may impact the battery. The reduced response time increases the amount of work that needs to be done by the device. All these calculations and the sensor's values gathering are expensive for the battery.

3.3 GAR Architecture

Firstly, let's explain the architecture. The encapsulation offered by the GAR ends up on a simpler architecture design. This is due to the default decisions made by the API that are already implemented, namely Sensors, Features, and Algorithm. Due to this the programmer basically only has to create the app, decide on what information is displayed, colors, and necessary buttons mainly. The big decisions and most important are already made by the API. Decisions that result in a specific transportation mode.

To assist us with the explanation of the architecture we have Fig. 3.3 that exhibits the structure of this solution. This solution is composed of 5 different components. The Main Activity, the Activity Adapter, the Writing Intent Service, the Activity Receiver, and finally the Activity Intent Service. The Main Activity is responsible for displaying all the information to the user. The Activity Adapter is responsible to provide to the Main Activity an updated list of each activity's confidence values. The Writing Intent Service is the service responsible for all the readings, calculations, and determinations of the current activity by calling the GAR. This is the component that sets the interval between two consecutive calls. It also writes the values on a log for further evaluation. The Activity Receiver is a Broadcast Receiver responsible to launch new Activities Intent Services. These are responsible to extract the results of the current

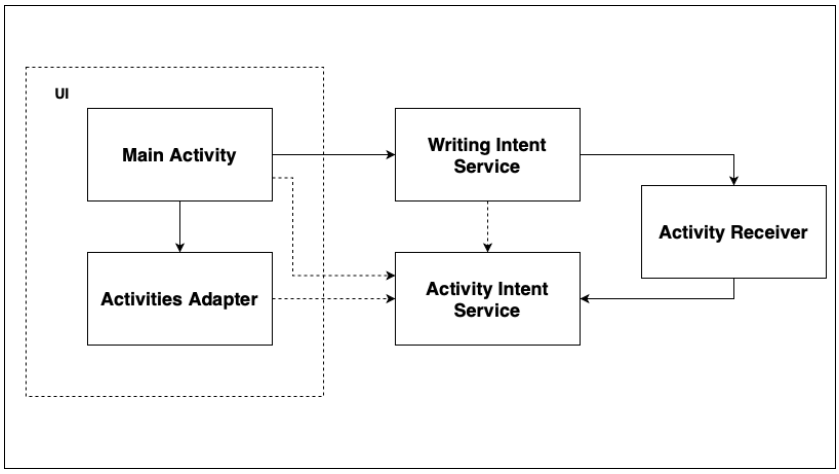

Figure 3.3: GAR architecture
Dotted arrows mean content access
Full arrows mean the creation of the component

activities and pass them both to the Activity Adapter, so it can pass them to the Main Activity, and the Writing Intent Service, so it can write them in the log.

Now that we have explained which function each component has, we need to explain their interactions. The Main Activity, the one that is launched at the time of starting the application, calls the Activity Adapter to refresh the displayed list of activities confidence. When the user starts tracking the activity, it launches the Writing Intent Service, which keeps running until the end of the tracking or app shutdown. The Writing Intent Service calls the GAR every x seconds defined by the user. It also launches the Activity Receiver, which launches the Activity Intent Service to treat the information provided by the API. It then passes it on to the already mentioned components.

3.4 GAR Algorithm

So now let's discuss the GAR algorithm. Here it is going to be explained/reminded how this algorithm works, what it does, and what it returns.

As explained previously (see Section 3.3) this is a simpler solution for the programmer. It is simpler to implement once this type of solution is mainly composed of one component, the GAR API. This component is responsible for determining which sensors are going to be used, get their values, select and extract their features and use the algorithm to classify them. All this is performed by the GAR so the programmer has to make fewer decisions reducing the work difficulty compared to other types of solutions. This is visible on the image (Fig. 3.4), where the cloud symbol represents the unknown information/ the decisions that are not taken by the programmer. This image displays the representation of the GAR algorithm applied on an app. As it suggests, the application just needs to call the API

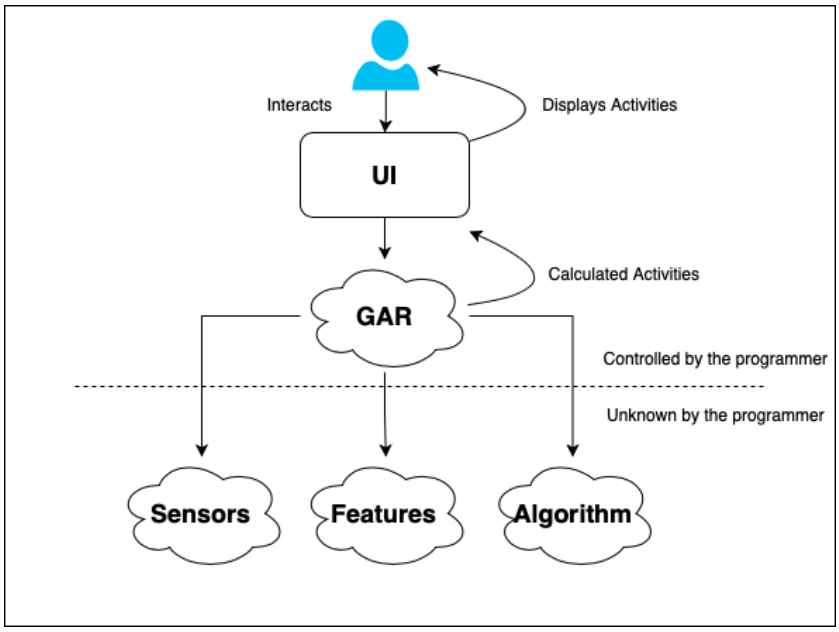

Figure 3.4: GAR representation
Cloud represents unknown info

and it takes all the necessary decisions to determine a transportation mode, specifically cycling for this work's purpose. The image also displays a dotted line separating what is controlled by the programmer and what is not. It is possible to observe that the Sensors, Features, and Algorithm boxes are not in the programmer's hands as was explained before in see Section 3.3. These three categories/steps belonging to a transportation mode detection application are also represented with a cloud because it is not explicit to the programmer which sensors, features, and algorithms are being used.

This algorithm is continuously running, apart from some exceptions presented previously in Section 2.4.2 when the GAR is discussed. A sampling interval is defined to determine the sampling interval, i.e., the time between two different readings. Each sampling interval performs readings, calculates features, runs them through the algorithm, and returns the probability of each one of all 8 possible detectable activities (IN_VEHICLE, ON_BICYCLE, ON_FOOT, STILL, UNKNOWN, TILTING, WALKING, and RUNNING). These activities were also presented before on the Chapter 2.

The probabilities vary between 0 and 100%. Only the activities with a current probability higher than 0 are returned. For instance, if at some time the result is 30% probability of walking and 40% of running, only the walking and the running activities have an assigned value. It is pre-defined that missing values equal a 0% probability. Those values are returned assigned to an id and not to the name of the activity. Each activity has a unique id assigned, being them IN_VEHICLE = 0, ON_BICYCLE = 1, ON_FOOT = 2, STILL = 3, UNKNOWN = 4, TILTING = 5, WALKING = 7, and RUNNING = 8. A possible return of the previous example would be something like 7: 30, 8: 40. The other values are all zero so are ignored and

are not returned by the API.

This probability classification is performed in each sampling interval but alone is not enough to classify an entire trip. This is because there is the question of which value to use? Do we use the last reading, see which activity has a higher probability, return it, and ignore all the others? A random reading and return the activity with the highest probability? This is why the current solution uses another algorithm over the GAR, but it is not considered for testing purposes. That is because here it is intended to compare directly the GAR with the RF once they can be used to determine an activity of this type. That way it is possible to assess if the GAR can be used as Google provides it using a simple algorithm over their results and at the same time compare between them which is better and presents better results. Instead, it is only made an average of the values to determine the activity during a certain trip. In other words, the activity that presents the highest average probability for the most amount of different readings is considered the performed activity. That is an attempt to reduce to the maximum the battery and time needed to get the result at the end of a trip to compare directly the GAR and the RF solutions.

3.5 RF Architecture

After analyzing the architecture and algorithm of the first solution, let´s analyze the second and proposed solution to replace the first. This solution as mentioned in Chapter 3 is based on the RF. For this solution, it was necessary a lot more work in terms of implementation and testing. This is due to the need to decide on all aspects relating to the determination of transportation mode. It was necessary to decide which sensors to use, which features were the most relevant, create and train the detection algorithm. These decisions take a lot of time due to the high number of possible sensors combinations but it was decided to use the GPS and the Accelerometer for this work based on previous works presented on Chapter 2. In particular, it was used, as a base model for this solution, the Woorti [27] application.

This solution is a bit more complex and has more components than the last one, as it is possible to observe in Fig. 3.5. This solution has 10 different components. The Main Activity, Trip Adapter, Trip Repository, Trip, Trip Service, Trip Manager, Util, Location Listener, Accelerometer Listener, and Classifier. The Main Activity, just like the other solution, provides the information displayed to the user. That information is obtained from two other components, the Trip Repository that has all recorded trips stored, and the Trip Adapter that orders them and passes them to the Main Activity. The Trip component is an object with the Trip specifications created each time a new trip is started. The Trip Service provides a service to start recording trips. The Trip Manager is responsible for the management of trips recordings, such as call necessary sensors, and stops them when the recording is finished. The Util component is just a component with useful functions that are necessary to use on other components, such as get the maximum of a list of numbers. The Location Listener is responsible for getting the GPS readings. The

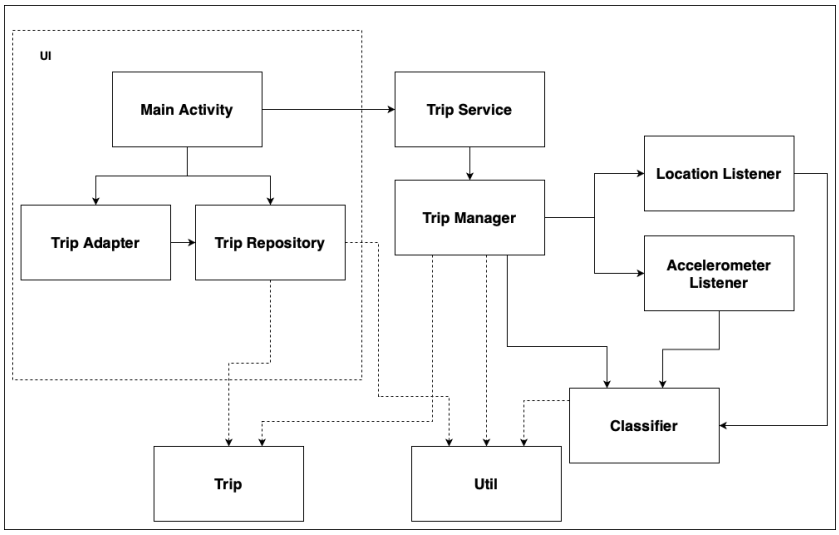

Figure 3.5: RF architecture
Dotted arrows mean content access
Full arrows mean the creation of the component

Accelerometer Listener is responsible for getting the accelerometer readings. Finally, the Classifier is responsible to classify/determine the recorded activity.

The application flow is the following. The Main Activity is created at the time of launching the app. This component makes calls to the Trip Repository and Trip Adapter to obtain the updated list of recorded trips. When it starts recording, the Main Activity launches the Trip Service, which is responsible to launch a Trip Manager. The Trip Manager is responsible to launch the sensors listeners, both Location and Accelerometer, in this case, starting recording the activity with that. The sensors Listeners are responsible for each x seconds get the values of the respective sensors and pass them to the Classifier. The Classifier stores the values in a log file for later analysis and comparison with the GAR application. When the application is shut down or it is pressed the stop recording button, the Trip Manager stops the Listeners and there are no more readings performed. After that, it communicates with the Classifier warning the trip record has stopped and it can now classify the trip. The Classifier runs the values obtained from the listeners through the RF algorithm and returns if the recorded activity was cycling or not.

3.6 RF Algorithm

Now it is time to talk about the RF algorithm. First, we need to clarify what this algorithm is and how it works. After that, the preferred sensors are named as well as the extracted features. Finally, briefly explain how it has applied to this particular case.

The RF is a ML algorithm that has as its base the DT algorithm. What this means is that, just like

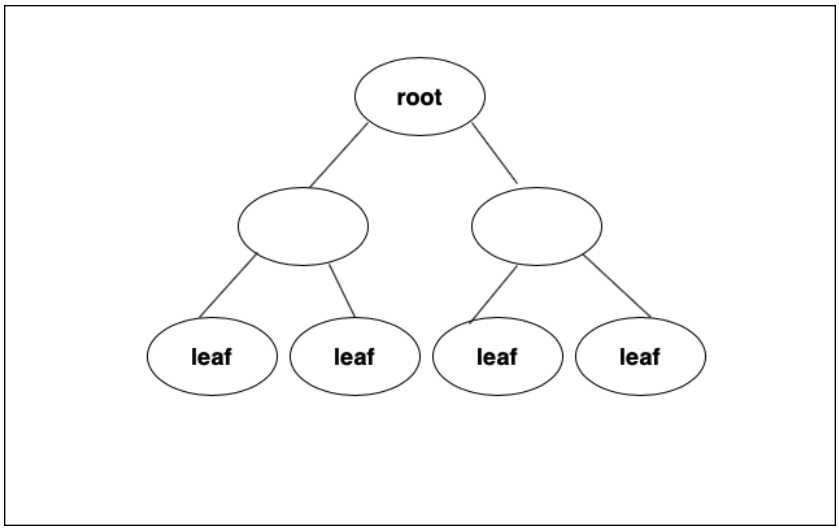

Figure 3.6: DT representation
Lines represent the branches
Circules represent the nodes

the DT, the decision is based on constructed trees. Those trees are composed of branches and nodes. There is a special node called root. This is the first node, the base of the tree. Each node has a decision to make and based on that it chooses to which branch the flow must continue. Those decisions are continuously made until the leaf node is reached. This is the last node of a tree, where there are no more branches to choose from and consequently no more decisions to make. For instance, let's imagine we have a tree composed only of seven nodes, one root, and four leaves (Fig. 3.6). Here there are two different levels, which means two independent decisions to make.

Let´s say that this DT aims to find if someone has a probability of getting cancer based on familiar background and blood pressure. To simplify let's say that if something checks at a node it follows the right branch, if not it goes to the left. The family background is passed to the first node, the root. It checks if this person has any family member who has or had cancer. It finds that he has so it goes to the right. In that second level node, it is checked if the blood pressure of that person is above or lower than a certain value. It says it is lower so it goes to the left branch. This is a leaf branch so there are no more decisions to make. This node returns the response to either that person is or not cancer-free risk. So this is a single tree and the RF is composed of several trees built in different ways. That is the random part. The more the number of features the more the number of different trees that can be randomly built. In this case, we could build a different tree where the root would check the blood pressure and the second-level nodes the family background. These alterations may end in a different result. The RF checks all its trees and stores their results and when all the trees have a result it sees which is most common and is decided as the final result. This algorithm tends to work better and has higher accuracy with the increase in the number of trees. This is due to the increase of coverage/checking of a higher

number of possible combinations.

Although, to reach this trained model, i.e., a set of trees configured to determine a specific result it is necessary to pass through a training phase. This is performed by gathering data and pass them through the algorithm, this is, the multiple trees and compare the result with the ground truth. In Fig. 3.7 it is shown a representation of the several steps necessary to perform before the model is ready to predict something based on raw data.

It starts by gathering data from the desired sensors, in our case from the accelerometer and the GPS. After that retrieve the selected features and label them with the ground truth. With this it is possible to compare the real activity (ground truth) and the model result. Only this way is possible to determine if the model is correct or not. Next, after labeling, the features must be run through the algorithm to calculate the predicted activity. Here the result must be compared with the one labeled to check the model's accuracy. The values used for decision-making on each node must be calibrated to increase the accuracy. This must be performed several times to ensure the model has tested a high number of cases/sensors readings.

At this stage, if there is the need to improve/calibrate the model it can either gather more and new samples from the sensors or use the ones already labeled and pass them through the algorithm again. This second option usually only occurs if the sampling data is already well-populated and diversified, just as it was mentioned in Section 1.2. After doing this several times and assure that the values used on the nodes to make a decision can not be more calibrated without harm the accuracy, the model must pass to its final phase, the detection phase. This is when the final model is obtained and ready to predicting current activities.

With the model ready to use, the flow of the algorithm is pretty simple. It is basically the same as the other solution, the GAR. The base is the same although they are not exactly the same. Fig. 3.8 shows the overall flow of this algorithm. As it shows and was said a couple of sentences ago, the base of this algorithm and the previous was basically the same. They both have the same three big steps in the same order, the sensors, features extraction, and the algorithm/model. The main difference is that here all this is chosen and made by the programmer while on the other was already part of the API. When the application starts recording the tour it starts reading the values of the sensors, the accelerometer, and GPS. It then extracts the necessary values and stores them. It also writes them in a log for further evaluation comparisons and cycling prediction in Chapter 5. This is performed while the trip is being recorded, no predictions are made unlike the GAR that for each reading is making a prediction. This solution only uses the model at the end, when it stops recording. At that moment, the sensors stop giving new readings and all the features are calculated based on the stored values during recording. Those features are then passed to the algorithm/model obtained from the training phase. It then builds multiple trees each different from the others with the features in different node levels to cover the highest

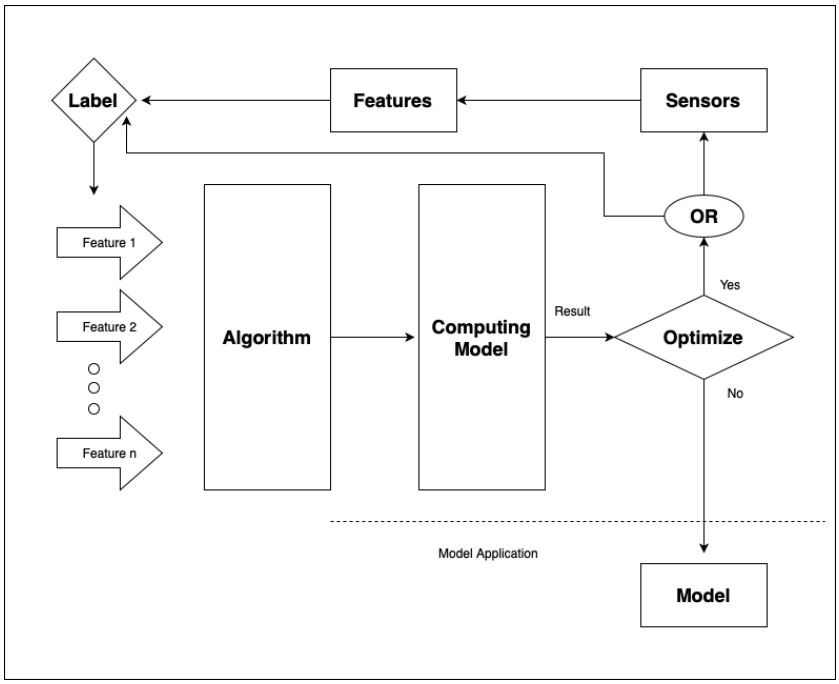

Figure 3.7: RF Trainig phase

amount of scenarios possible. It passes the values through them obtaining an activity result from each one. Here the result is either cycling or not. It then compares those results to obtain the one with the most votes, resulting in the final verdict. This is the only value that passes back to the user and he can see it on the application User Interface (UI).

3.7 Summary

Once the main purpose of this work is to study the best way to detect when a user cycles, using only mobile phone sensors, there are several aspects to consider. Both solutions are local, which meets the second requirement, the low response time, once it reduces the communication between devices, having no communication at all. Not being dependent on external factors, such as communication problems between devices or problems with external devices that may take longer to answer. It is necessary to take into account that mobile phones work on batteries, and those are not infinite. So the app should not consume all the phone batteries when used. Also, it is important to remember that the user needs his phone not just to use the app but also for other apps, for instance, Google Maps to get the route to a meeting or to call someone. These limitations meet the third requirement. The fourth requirement is fulfilled by implementing the app to Android, while the first requirement is met by refining both solutions to achieve the highest possible accuracy.

It is discussed the design of both solutions as well as how they work. The GAR is a much programmer-

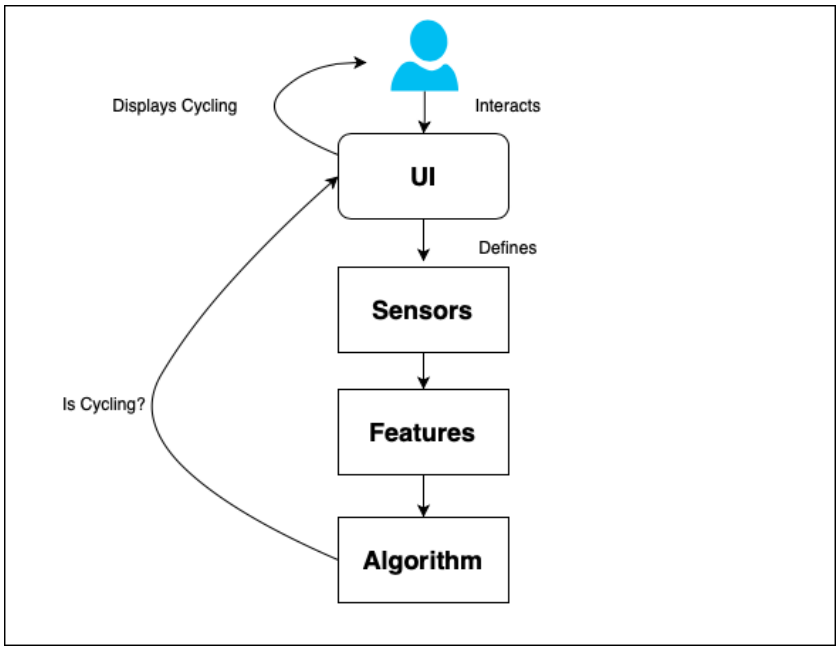

Figure 3.8: RF prediction phase

friendly solution once it is simpler to use. It performs all the difficult work, i.e., all the work related to determining the activity. Because of that, the solution becomes lighter and fewer components and instructions need to be made. The RF on its turn, is way more complex when compared to GAR. This solution requires that the programmer makes all the decisions on sensors, features, and the algorithm. This results in much more work for the programmer once all those decisions need then to be coded. The worst/more arduous part of it is the algorithm. The algorithm is hard to implement and it needs to be trained. That training requires a lot of time to gather the necessary readings and then use them to train the model. That training must be done until the highest accuracy is achieved. All this work results in a higher number of components on the architecture.

4

Implementation

Contents

In this chapter, it is discussed the implementation performed for this work. It starts by describing the implementation environment. This aims to introduce the applications and languages used to implement both solutions. It has two main themes of discussion. The first is the GAR implementation performed to obtain the values currently obtained by Biklio. The second, and the suggested alternative by this work, is the RF implementation. Some implementation challenges/particularities are also addressed here once they can really impact the normal/desired app functionality/performance.

4.1 Implementation Environment

Here is described the implementation environment. It describes all the necessary tools used during the development of these applications. It also explains briefly how they work and why they were chosen for this work.

First of all, let's start with the main condition of this work. As mentioned in Section 1.2 as the fourth and last requirement, the Android was chosen as the target OS where these apps should run. It was decided to choose it over the iOs, which is the Apple OS. This decision was made based on the available programming languages for each OS. In other words, it was necessary to choose a programming language that was able to be used to implement both solutions. A language that had the necessary libraries and resources to be used for determining an activity. Android owns one of the most well-known and common programming languages. Naturally, it conditioned the rest of the decisions made after. Once chosen the target OS, there were some limitations/restrictions that appear. The first and most obvious was that the applications could not run on any Apple device. They would only work on Android devices. The second was the existing languages.

Nowadays there are two main languages for Android programming. The first and oldest is Java, and the second and more recent is Kotlin. They are both capable to implement a solution of this type. They both own several libraries that allow us to implement multiple algorithms. For this case, in particular, which aims to implement the RF algorithm the languages must be compatible with it. This happens for both of them, owning both a couple of libraries capable of implementing the RF. Although the decision here was based on the accessibility of content and familiarity. In these chapters, Java wins because it is older so it has more content on the internet making it easier to find bug fixes/corrections. Also, due to its age and time on the programming world is a more well-known language for most programmers. Java owns some pre-built machine learning libraries, such as WEKA [44], JDMP [45], and MLib (Spark) [46]. Another advantage of Java is that it is capable of scaling to larger systems or applications due to being a general-purpose language built for cross-platform development [47]. Finally but not least, once the app is going to be tested in android it is needed to use an appropriate programming language and Java is the oldest official language.

Both solutions were developed on Android Studio. That is an app that allows the programmers to emulate the Android system and even an Android phone without requiring a physical Android phone. This feature is really interesting once it allows the programmer to develop all the applications without extracting them from the computer. This platform allows emulating several aspects of the system as the rotation of the phone and some others. The programmer can select which Android version he wants to test. This way it is possible to test for multiple versions without actually owning any of them. When the applications are ready, i.e. tested, they can be passed to the user's phones and start tracking their activities so they can know if their activity was recognized as cycling and start earning discounts/offers.

4.2 GAR

For the GAR solution implementation, it was created an app. That app displayed all the possible transportation modes detected by the GAR and their confidence degree. This was performed based on readings made from the sensors every x seconds. Those readings are then treated, calculating the selected features, and passed through the algorithm returning each activity's probability/confidence. Those results are stored in a log for further analysis. This log is used at the evaluation time to help to find wrong transportation modes and some other cases explained in Chapter 5. The confidence levels are displayed in real-time so the user can see which mode has a higher confidence level and which modes are detected at the time. This process repeats itself every x seconds and until the user stops the application. This type of configuration facilitates the evaluation stage once the current transportation mode can be compared in real-time with the real transportation mode.

Its UI, visible in Fig. 4.1, is very simple once this app is used only for testing. It is composed of 5 buttons and displays the 8 possible modes detected as well as the sampling interval. All its information is presented on one screen. The initial screen is the only one that exists and the user sees, to minimize the complexity to the maximum. The first three buttons are used only to build the ground truth, i.e., write on the log the current activity. These are mainly focused on the Evaluation part and are very important there as well because those records indicate the start of an activity/transportation mode. Each time the user starts a new activity, one of the three options, presses the corresponding button and that activity name is recorded on the log with a timestamp. So, looking at the log, it is possible to know that the values after a certain activity type are all for that type of activity. With that, it is easier and possible to know when the app is returning activities that were not performed at the time of sampling. The other two buttons have a direct impact on the execution of the app. One is used to start tracking the activity. The other is used to define the sampling interval. This last button only affects if the start tracking was not pressed yet. This is because we do not want different sampling intervals in the same test. The sampling interval is the number of seconds that the readings must be performed. It was called previously x. It

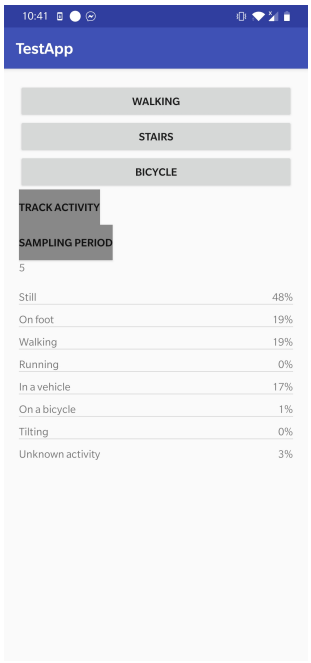

Figure 4.1: GAR user interface

is the value that defines the time between two consecutive readings. For this purpose, the sampling intervals accepted were 5, 10, 30, and 60 seconds. These values are the only values the user can select to track his activities. Finally, the app presents the 8 types of activities detected by the GAR. In front of each one, it presents its confidence level.

To perform those readings it was used a background service called Job Intent Service. As the name suggests it is a service that runs in the background, i.e., runs even when the screen is turned off. As explained before there are two different types of APIs, the Activity Recognition Transition API, and the Activity Recognition Sampling API. The first is more strict giving less power of choice to the programmer where the second allows it to control more things such as sampling intervals. Based on that it was used, for this work, the second option to be able to compare and evaluate different sampling intervals impact on accuracy and battery. The user, by pressing the start tracking button, requests for activity recognition updates. That is performed calling the requestActivityUpdates() function [8]. This function handles everything and returns confidence values for each detected activity. These are the values that are saved on the log and displayed to the user. To ensure that the latest version of the GAR was being used it was necessary to update the play-services-location library. It was used the ”com.google.android.gms:play-services-location:17.1.0” because it was the latest version at the time of creating the app.

4.3 RF

For the RF solution was created a different app. This is since this type of solution requires other development steps. Both apps have the same bases, i.e., both apps read from sensors and use their values to calculate the final activity and display it to the user. However, they have different implementations as is explained next. This app shows to the user the end result, i.e., if the recorded trip was cycling or not. Similar to the GAR application, readings are being performed each x seconds and stored as well in a log file to create graphics and be able to evaluate the solution. Here start the implementation differences. This kind of solution needs the programmer to choose which sensors must be used. As well it is encharged to chose which features must be extracted from each sensor. With those features, it must run them through the RF algorithm to determine the activity. This part is divided into two steps.

The first is the testing phase where, as the name suggests, the algorithm is trained and refined to obtain the most accurate output possible. This is performed by passing the features extracted, with a ground truth attached, through the algorithm and compare its result with the ground truth. A feature with a ground truth attached is saying that for a certain set of features they have already a determined activity associated. This is performed previously, by matching each set of features to a ground truth activity after recording some trips doing some activities. This action is called labeling. The higher the number of times the result matches the ground truth, the higher the accuracy of the algorithm. With higher accuracy, it means that the adjustments necessary to improve the algorithm are less significant than if the algorithm presents a lower accuracy. This phase is really important because it prepares/trains the algorithm for the next step.

In this particular case were used 22 different features. They are mostly of three different types, acceleration, speed, and position. Some of the values are filtered before being passed to the model to reduce the error. The features used were avgAccel, minAccel, maxAccel, stdDevAccel, avgFilteredAccel, accelsBelowFilter, accelBetw_03_06, accelBetw_06_1, accelBetw_1_3, accelBetw_3_6, accelAbove_6, avgSpeed, minSpeed, maxSpeed, stdDevSpeed, avgAcc, minAcc, maxAcc, stdDevAcc, gpsTimeMean, distance, and estimatedSpeed. They were stored in a HashMap relating the feature name with the respective value. The raw data provided by the sensors, accelerations, and locations namely, were stored in lists.

The second step is the model application phase, i.e., the testing phase where the algorithm/model is used to determine an activity based on features that do not have a ground truth attached. These features are obtained in real-time, unlike the ones used in the training phase, that were previously obtained, labeled, and passed to the algorithm. Here the features are passed without pre-labeled ground truth. The result of the algorithm is considered the ground truth for the app. This is the reason why the first phase is so important, a good training phase results in a reliable model application.

The UI here is different from the GAR because there is no need to show and calculate all activity

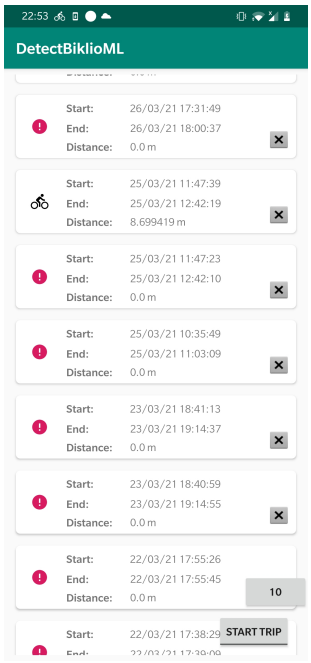

Figure 4.2: RF user interface

types. We just calculate the cycling activity once is the aim of the work. So it is only presented to the user if a trip was or not detected as cycling (Fig. 4.2). The application does not have the same three buttons as the GAR app. During the testing of the other app, the other buttons were not as useful as was initially thought. That was because the trips were performed by activity type, mostly cycling, and the transition between different activities was not the main goal of this work. So for that reason, the transition was not tested. Once this app was built after the other those buttons were not implemented. To distinguish between the activities first, it was tested one, and then after the results were extracted test the other activity. Just like the GAR UI has a button to define the sampling interval for testing reasons. The sampling intervals accepted are the same as the GAR, 5, 10, 30, and 60 seconds. This once again is for evaluation purposes that are described later in more detail. This application has a new feature the previous one did not have. This one saves the last trips and displays them to the user. In case the trip was determined as cycling it shows a bicycle image on the correspondent trip, otherwise, it presents an exclamation point in a red background as it is observable in the image. Finally, has the start/stop button that starts or stops tracking the activity.

This solution, just like the previous, has a service performing all the readings, writings, and final activity calculations. This one has also a service that is always running even when the screen is turned off. This is a key feature for applications of this type once they require to be collecting samples from the sensors and store them constantly.

4.4 Challenges

During the implementation of these applications, we have faced some challenges in order to get them to work as intended. This is not strictly necessary but it may help for future works, similar to this that intend to perform an action multiple times within a specified interval. It can help someone to avoid these challenges and save time by not having to deal with them. Firstly we must provide some context so those challenges can be easily understood.

This work, as explained previously, aims to detect cycling activities. To do that it reads from the sensors, writes/saves the results, and calculates the transportation mode, resulting in cycling or not for this particular work. These actions are performed every x seconds, being those defined by the programmer. The collection of multiple gathered sensor values, at different times, provides the necessary knowledge to indicate if the activity was cycling or not.

The first challenge arises due to the necessity to gather values from the sensors each x seconds. There are a few ways that this can be made on Android. The main one is the Service class [48] but not all services can be used for that. The most suitable at a first look would be the Background service. This runs in the background even when the screen is locked. The Intent Service was a background service that fulfilled those requirements. It sounds like it would work but since Android API 26 (Android 8.0) this type of service has suffered some serious restrictions from the android system to save battery. The result was that the system would only give around ten minutes for the service to run and it would stop it after that, and would not launch a new one to replace the stopped one. It would resume/start a new service after some minutes, depending on the inactivity on the screen. If the user was constantly unlocking the screen it would have smaller intervals of stopped readings than if he unlocks it occasionally. So we changed to the Job Intent Service which was a new version of the Intent Service that handles better the restrictions imposed by the Android system and for older APIs, it behaves similarly to Intent Service. Although it presented the same problems as the previous solution. After a while, the system would stop the service and would not launch immediately a new one to replace the old one. So it was tried to use locks to gain the CPU access as an attempt to keep the service running but that did not result as well. It only kept running if the device was charging or the screen was turned on. And these are not viable solutions because the normal user does not have his device charging every time he rides and uses the app. He also does not have his screen always turned on during a ride. This last one goes is the worst one because it goes against one of the requirements, the third one. So other solutions had to be used.

The Foreground Service was used once, after some research, it was said that it handles the restrictions imposed by Android after API 26. It was tested with different implementations, locks, local broadcasts, and normal broadcasts when the service stopped. None of them worked. It was tried to set a timer smaller than the ten minutes timeout (supposedly) imposed by the system to terminate the service and launch a new one but it did not work as well. It was tried with broadcasts as well and for the

Background services but none of them with success.

After more research and looking through one of the phones it was found an option to disable battery optimization for a specific app. After checking that box the app was tested with the foreground service and the background service, Job Intent Service more precisely, and for both solutions the app worked as intended, gathering samples and saving them each x seconds. Both without any extra code such as locks, timers, or broadcasts. This is a much simpler and the only solution that really works for this purpose. This could also be achieved using code instead of manually although this solution does not work for all Android devices. For some, it only works after the app was shut down and launched again. There are also some Android models where this never works. So the code solution is probably not the best to use yet while these limitations exist and it is only available from API 23 (Android 6.0). To call that it was necessary to use the Request_Ignore_Battery_Optimizations permission, displayed in Appendix A.

The last difficulty was that although all the devices used for testing were Android they were from different brands and each brand does its twists over the Android base, resulting in slightly different systems. In other words, the same API version may not be exactly the same between two different phones from different brands. They can respond differently to the same functions.

4.5 Summary

As seen in this chapter, the GAR solution is much simpler and easy to implement than the RF. It does not require any sensors implementation or features selection and calculation, or even algorithm implementation and training. Also, the implementation of an application that uses GAR is mainly focused on the visual context and not so much on the application goal, determining if the user is cycling. It only has to call the API and it performs all the work. It is just needed to present the results. On the other hand, the RF needs all those steps to be performed by the programmer. That results in a longer implementation process mainly due to the algorithm coding and to its training.

5

Evaluation

Contents

For evaluation purposes, it is intended to compare the GAR with the RF. This aims to compare both algorithms and determine the best, with a special focus on cycling detection. By comparing both GAR (current) and RF (suggested) algorithms it is expected to understand better how each one behaves for this particular case.

It is going to be explained the testing environment for the testing phase. All the measurements took during the evaluation of both solutions, as well as the conditions and rules applied. It is also shown a comparison between the two algorithms. The comparison is performed aiming at multiple aspects. Those being the requirements described in Section 1.2, namely accuracy, time response, battery, and finally OS. What this means is that both algorithms are compared by each one of these requirements, besides from the OS which is the same for both (Android). Both algorithms were only tested on Android devices, once both apps were written in Android. None iOS device was used to test these algorithms due to incompatibility software. More specifically because the programming languages are different for both Android and iOS.

5.1 Testing Environment

Let's start by discussing the testing environment, i.e., where the tests have been performed and under what circumstances. This section contains all the explanations and conditions under what the testing phase was performed. It describes which phones were used, along with their main features, battery size, current OS, Random Access Memory (RAM), Read Only Memory (ROM), CPU and year of release. These features can be important to understand whether any of these features may impact the end result. The year may be the least significant however is shown because normally the older the device the slower it is. Also because there is a certain limit of updates a phone can make with a certain set of specs. A phone that is put into the market from the factory with Android 6.0 hardly gets an upgrade to the Android 10 or 11 most probably because it is outdated and their brand stopped providing updates for that specific model or because that model specs are not compatible with newer versions of Android. It is also described placement positioning. This says where the devices were placed at the time of testing. It is important because it explicitly says the tested positions and that on any other position the results may differ.

For the testing phase, there were used three different phones, from different generations, brands, sensors, and versions of them. They also have different versions of the OS running. These differences are very important for testing purposes, acknowledging that compatibility is one of the main goals of this work, i.e., the app should work for the largest amount of phones. Different phones have different hardware and android versions. So the bigger the number of tested phones the better, helping to improve the system accuracy. To notice that the natural evolution of the technology is to continuously add new

versions, so the most important versions are the most recent ones. This is due to the technology's natural evolution. With new technology, older methods are deprecated and replaced by new and improved ones. The user normally follows that evolution by buying more and more recent phones, decreasing the number of used old phones (with the firsts android versions) more and more until it comes to near zero. For instance, currently, the percentage of android phones running android 7 or higher represents something like 87% of the total Android devices [49]. Android 6 or higher represents about 93% of the global share. This means that, nowadays, only around 7% of all android devices are running Android 5 or lower. This data supports the previous idea that the newest versions are being chosen over the firsts.

The used phones were the following, Oneplus 7, Oneplus X, and Xiaomi Redmi Note 4. The first is the most recent of the three. It was released in June 2019, runs the Android 10 with a 3700 mAh battery. It has a 256 Gigabyte (GB) ROM, a 8 GB RAM, and an Octa-core (1x2.84 GHz & 3x2.42 GHz & 4x1.78 GHz) CPU. The second is the oldest of them, being released in November 2015. It is running Android 6.0.1 and has a 2525 mAh battery. Owns a 16 GB ROM, a 3 GB RAM, and a Quad-core 2.3 GHz CPU. Finally, the Xiaomi was released in January 2017 and runs Android 7 with a battery capacity of 4100 mAh. Owns a 32 GB ROM, a 3 GB RAM, and an Octa-core 2.0 GHz CPU. The battery capacity is shown because the battery impact that each approach has was also tested and explained later on.

Each approach was tested for two different positions. One was in the front pocket of the pants and the other was in a backpack. These two placements were chosen because these were the most natural and probable positions where a user would place his device during a daily cycling activity. The solution was tested during cycling but also going up/downstairs. This was because the movement of the legs when going up/downstairs is similar to when cycling. Both present a more rounded movement seen from the side. This was an attempt to detect possible false positives due to those similarities of movement.

To avoid/minimize the differences between cycling trips, the route performed during the testing phase was always the same. The reason why is because we can not test the two solutions on the same phone at the same time, so it was decided to take the same route as an attempt to simulate that the same conditions were present for both solutions. The same happened for each sampling interval pre-defined of both algorithms. Finally, it must be explicit that the route was mostly out of the center of the city, avoiding many traffic lights. This means that during testing the cycling activity was most of the time a continuous movement once there was not the necessity to stop at many traffic lights.

For each algorithm and phone position were obtained more than 10 hours of cycling samples for each phone. This all summed up means that for each algorithm and phone placement were gathered 30+ hours. Once there are two different placements for each algorithm it means 60+ hours for each algorithm and a total of 120+ hours of testing.

With the cycling testing, we can detect each algorithm's false negatives, i.e., the number of times a user is cycling but the algorithm does not recognize his cycling. But it is also important to test the

false positives, this is, the number of times a user is not cycling and the algorithm determines that he is. This is important to assure that the app does not offer prices/discounts to users that are not cycling. So the activity of going up and downstairs was tested as well. This choice comes from the fact that the movements of this activity are similar to the ones performed cycling, as already mentioned. This way it is intended to understand if the algorithms would recognize it as cycling.

5.2 Accuracy

When it comes to accuracy, the evaluation phase was divided into several steps. For this evaluation, some aspects need to be considered. The first was the phone placement, i.e, where the phone was placed during the activity tracking could impact the accuracy results. This is because the movement the phone has while in the pants pocket is not the same as when in a backpack. The second was to compare each tested activity, cycling and up/downstairs, to determine their accuracy and possible false positives or false negatives, as already mentioned. It was also necessary to evaluate each phone individually to understand if the algorithm performance was dependent on the phone model or not. Finally, was necessary to distinguish between sampling intervals to comprehend which presented a higher accuracy. To present the results of these tests was decided to use tables. Those tables relate the phone model with sampling intervals, at the edges. The values in the middle of the table represent the accuracy obtained. This results in four tables for each algorithm, one for each device placement and one for each activity.

There is one big difference between these two solutions when it comes to accuracy evaluation. The RF can only determine an activity when it ends. The GAR can do the same by the end seeing which activity had a higher probability during the trip. Although, once this solution tells each activity's probability each time it reads from the sensors, it can also perform an individual evaluation. It would consist during only cycling or only up/downstairs activity recording to see if all the returned values are compatible with the activity being performed. This solution would require fewer trips, once in one trip, it is possible to evaluate the accuracy percentage while on the RF solution one trip only returns one value. In other words, the GAR provides for instance 100 readings in one trip, resulting in 100 activities returned. Where the RF, in the same trip, provides 100 sensors readings but only one activity. To have the same amount of values on the RF it would be necessary 100 trips. The GAR was tested and measured its accuracy by making the average of all cycling values during a trip, removing the initial and final moments. These moments are relative to the time the user presses the start tracking button, puts the device in place, and starts cycling. Those moments were not considered for the RF as well.

Before starting analyzing both solution's accuracy, let´s observe for a moment the reason why the GAR needs to have an extra treatment/algorithm. In Fig. 5.1 it is displayed a small part of a cycling trip. The image presents all the 8 possible activities the GAR detects. Each is represented with a different

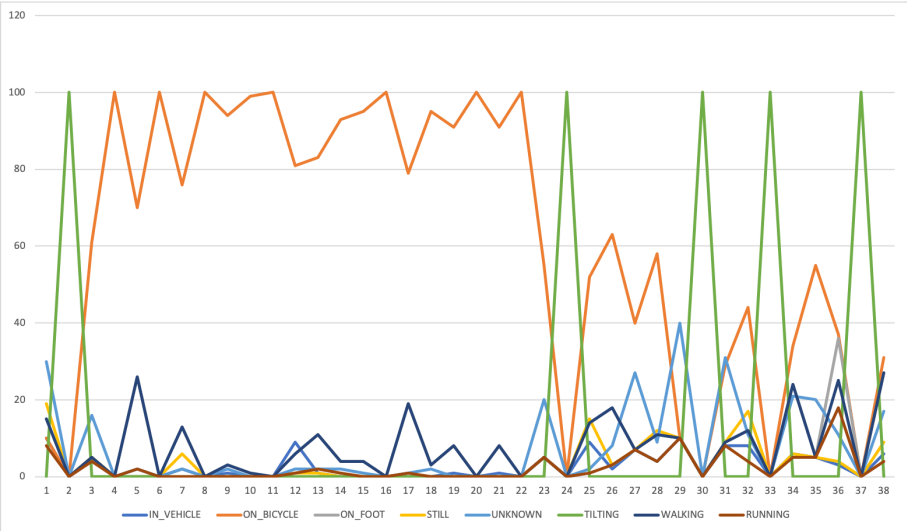

Figure 5.1: GAR cycling output example

color and labeled at the bottom. In there it is possible to see that although the user is cycling the results are not always matching that. At certain moments the activity with higher probability is not cycling but is Tilting at moment 24 or 30 for instance. At moment 29 it detects the user's most likely activity is Unknown. If no other algorithm was applied and it was decided to keep the last value as the trip value and if that value was not cycling that user would not benefit from his cycling due to an application error.

Moving on to GAR's evaluation we started by testing its accuracy while cycling. For that when the trip record button was pressed the user/tester started cycling and continue for 30 minutes at least. Few stops were performed during the testing phase, only the necessary due to traffic lights or to stop recording were made. This made that most of the time, over 90%, the user was cycling so the outputs are around 90% relative to continuous cycling. As mentioned previously that was performed for both front pocket and backpack positions. Table 5.1 and Table 5.3 show the results obtained from those tests for the front pocket for cycling and stairs respectively, while Table 5.2 and Table 5.4 for backpack. For all tables, the values on the vertical left point the used phones, and the top horizontal mark the sampling intervals. Although, while in Table 5.1 and Table 5.2 (tested while cycling) the higher the values the better the accuracy, in Table 5.3 and Table 5.4 (tested while on stairs) the lower the better. That is due to the objective of these last two tables. They are used to evaluate the up/downstairs movement and are tested to see if the cycling activity is detected. So the fewer times that happens the better, because it means fewer false positives.

	5	10	30	60
OP7	84	85	80	76
OPX	84	82	83	79
REDMI4	81	85	82	77

Table 5.1: GAR cycling accuracy front pocket
First line has the sampling intervals
Other lines display the accuracy (%)

	5	10	30	60
OP7	80	83	77	74
OPX	86	82	81	79
REDMI4	78	82	76	72

Table 5.2: GAR cycling accuracy backpack
First line has the sampling intervals
Other lines display the accuracy (%)

Starting by analyzing the GAR cycling results (Table 5.1 and Table 5.2) it was possible to determine that the solution presented better results for smaller/median sampling intervals. The bigger interval, 60 seconds was for all cases the worst one. This can be since with a larger interval in the same amount of time there are fewer readings, which means a higher probability of a set of wrong readings influences negatively the final result. Once the decision of the determined activity is based on individual decisions of the GAR and each of those individual decisions is taken at the time of reading if the reading is performed while the user is stopped at a light signal or for any other reason that specific decision determines the user is not cycling. These types of situations along with others similar can corrupt the final decision. While the smaller samplings have more readings to mitigate the misreadings caused by a set of bad/incorrect readings. The front pocket presented slightly better results overall than the backpack. That might have been related to the fact that in the front pocket the phone moves more than when in a backpack.

	5	10	30	60
OP7	6	4	4	3
OPX	7	6	5	6
REDMI4	5	4	5	3

Table 5.3: GAR stairs accuracy front pocket
First line has the sampling intervals
Other lines display the accuracy (%)

	5	10	30	60
OP7	5	3	4	4
OPX	6	6	7	5
REDMI4	4	4	3	3

Table 5.4: GAR stairs accuracy backpack
First line has the sampling intervals
Other lines display the accuracy (%)

In terms of false positives, i.e., the number of times the cycling activity is detected while going up/-downstairs was low for all devices and samplings. As displayed in Table 5.3 and Table 5.4 the percentage of false positives for this algorithm does not goes higher than 7%. This means that on average, 7 out of 100 up/downstairs activities are detected as cycling, and this is the worst case. The best case is around 3%. As the tables show, the values are similar between device placements with a smaller error (percentage of false positives) in the backpack position. That may be related to the fact that in that position the phone is moving less than when in the front pocket. The device's results were also very similar, not having a large difference between their values. Finally, it should be noticed that the percentage is almost leveled throughout the sampling intervals. This is, the values do not seem to depend/vary with the sampling interval change. They tend to be all-around a certain value, no matter the chosen sampling.

Moving on to the RF the exact same tables were constructed. Table 5.5 and Table 5.7 show the results obtained from the tests for the front pocket for cycling and stairs respectively, while Table 5.6 and Table 5.8 for backpack.

	5	10	30	60
OP7	89	91	91	89
OPX	88	90	92	89
REDMI4	89	92	92	90

Table 5.5: RF cycling accuracy front pocket
First line has the sampling intervals
Other lines display the accuracy (%)

	5	10	30	60
OP7	88	90	91	89
OPX	87	88	89	86
REDMI4	89	93	91	89

Table 5.6: RF cycling accuracy backpack
First line has the sampling intervals
Other lines display the accuracy (%)

In Table 5.5 and Table 5.6 are represented the cycling accuracy obtained for both placements. In this case, unlike the former one, the results are more well-distributed between all sampling intervals. Here the larger sampling results are closer to the results obtained on the smaller samplings. Also for both placements, the results are better overall for a considerable amount. This can be related to the way this solution works. Unlike the past solution, it does not predict the activity each time it reads from the sensors. It only does that in the end, when there are no more reads and consequently no more new values to consider. This might be important because it means the solution considers the entire trip readings to determine the activity and not a single reading. This way the bad/defective readings may be mitigated because the other correct readings are correcting the final value. To explain it better it may be better to present an example. For instance, suppose a user has stopped one minute in a light signal. The previous solution, depending on the sampling would return either 12, 6, 3, or 1 results of STILL. Here the accelerometer values might return 0 the same amount of times because there was no movement during that time. Although this solution has into consideration the prior and next moments and if in those the values indicate the user is cycling the impact of those bad readings does not affect as much as the others in the GAR. Likewise, the difference between placements here is almost none. Both present pretty similar values.

	5	10	30	60
OP7	1	1	1	1
OPX	2	1	1	2
REDMI4	2	1	1	1

Table 5.7: RF stairs accuracy front pocket
First line has the sampling intervals
Other lines display the accuracy (%)

	5	10	30	60
OP7	1	1	1	1
OPX	2	1	1	2
REDMI4	1	1	1	1

Table 5.8: RF stairs accuracy backpack
First line has the sampling intervals
Other lines display the accuracy (%)

The same happened when it comes to false positives. This solution, when tested up/downstairs rarely presented the cycling activity as the determined activity. That is possible to confirm in Table 5.7 and Table 5.8. Just like in the cycling accuracy determination, here the results are almost the same for both tables no matter the device placement. Results those that are low and show that the probability of an up/downstairs activity being confused with a cycling activity is lower than when used the GAR.

Finally, it was briefly tested cycling on a statical bicycle. The main reason why this test was performed was to check if the GAR solution would recognize it as cycling or not. The RF was also tested to guarantee that it does not return any cycling activity. That is because it uses the GPS and the user is not changing his position through time, so it is expected not to return cycling. For this work, it was decided

that this type of activity should not be detected as cycling, although the user is cycling in a statical bicycle because this is mainly for fitness purposes. It does not meet the main requirement/goal of Biklio, the CO2 emission reductions by using the bicycle as a transportation mode. During these tests, as expected and predicted the RF did not determine the activity as cycling while the GAR did, mainly when the user was not seated while cycling. This test increases the number of false positives that can occur for the GAR. This was not extensively tested, it was just a few minutes on the statical bicycle but it is worth being mentioned due to Biklio's main goal.

5.3 Response Time

Once both solutions are local, i.e., perform all the calculations in the phones and do not depend on any other device, both solutions present a fast response time. This decision was made to meet one of the requirements of this work as explained previously. This requirement, response time, was tested by calculating the exact time the user needs to wait to obtain the result of the determined activity. In other words, the time difference between the moment he stops cycling and he sees if the tracked activity was detected as cycling or not so he can use his benefits. For that, when the app stops recording the current time is saved. That allows knowing the exact moment when the activity ended. After the algorithms returned the detected activity it was gathered the current time as well. This second was to compare with the first and get the difference between them. The results are visible in Table 5.9 and Table 5.10 for GAR and RF respectively. The times presented in the tables are in milliseconds.

	5	**10**	**30**	**60**
OP7	57	47	36	29
OPX	72	58	53	42
REDMI4	68	55	49	37

Table 5.9: GAR average time
First line has the sampling intervals
Other lines display the response time (ms)

	5	**10**	**30**	**60**
OP7	531	510	487	459
OPX	612	603	596	589
REDMI4	587	580	569	554

Table 5.10: RF average time
First line has the sampling intervals
Other lines display the response time (ms)

Here the same trip was tested, which means the time between the beginning of the trip and the end of it is approximately the same for all devices and algorithms. It was chosen the same trip to remove all external variables that could corrupt the final result, distinguish only the devices and the algorithms. For instance, a possible external factor of the usage of two different trips in two different devices or algorithms is the number of readings. At the same amount of time, there are the same readings, approximately. If one trip has more readings than the other the response time result would not be reliable. That is because there would be more values to analyze and calculate for one trip than the other. This would result in a different response time, not because of the algorithm/device power but because the input data is bigger for one case.

As it is shown, the bigger the sampling interval gets, the faster is the response. This is a common

behavior for all devices. The same happens with the algorithm used. This can be due to the reason that in the same hour of recording, each sample has a different number of readings. For instance, in an hour the 60 seconds interval performs 60 readings while the 30 seconds interval performs 120. The 10 performs, even more, ending with 360 readings to be analyzed and calculated while the 5 seconds interval performs around 720 readings. This is 12 times more data to analyze than the 60 seconds interval. Here the GAR has an advantage once it displays lower response times. In fact, the difference between the GAR and the RF is in the order of 10x or more. It is a pretty big difference, although, it is necessary to remember that these values are in milliseconds. What this means is that the worst-case displayed here, which is the RF solution for the OnePlus X (OPX) at a 5 seconds sampling and it returned a 612 milliseconds response time. That is 0.612 seconds. This is fast enough to return the determined activity to the user after he stopped tracking it without him noticing he has to wait for the result.

It is important to remember that these values are only relative to the moment the trip ends until the returned activity. The RF performs all the calculations at this moment, performing only readings before. While the GAR for each reading performs calculations to determine the activity and in the end, the tested response time, all it does is sum those cycling values and divide them by the number of readings obtaining the average. This is way less complex than pass the sensor's values through the model and determine the activity. If the calculations performed during the trip recording were counted and added to these values the response time would increase.

5.4 Battery

Before getting into comparisons of battery performance between the two solutions some aspects of batteries should be clarified. It is important to understand/ consider that apart from the different battery capacities, the screen sizes and the number of pixels also play an important role in its lifetime. Also, it is necessary to know that currently phones own lithium batteries. Those batteries are claimed to handle 300 to 500 charge cycles. A charge cycle is a complete charge. For instance, from 0 to 100% is one charge cycle. While from 30 to 80% is half of a charge cycle. With the increasing charge cycles, the battery's capacity reduces gradually [50]. What this means is that an older/higher charge cycle battery has less capacity, which means that the time to decrease 1% of its capacity with a higher charge cycle may be less than when it had a smaller charge cycle. For the testing phase, the battery percentages used were approximately between 60% and 80%. This is to try to preserve the battery lifespan, once it is studied that the better interval for batteries is around those values. In fact, the previous reference shows that batteries kept between 65% and 75% showed much better capacity retention than the battery charged between 100-25%. Also, [51] says that the mobile phone battery should be kept between 20 and 80%. They should not be regularly charged to 100% because it is claimed that lithium batteries

work better in the middle.

Now that the normal battery behavior is explained it must be discussed the battery performance of each solution. In terms of battery, the values of consumed battery do not vary with the position changing as it occurs with the accuracy. It only depends on the sampling interval. The smaller the sampling interval the higher the number of readings. The higher the number of readings, the higher the energy consumption. On the other hand, the higher the sampling interval the lower the energy spent on gathering samples which means less battery consumption.

After analyzing many hours of sampling for each solution it was possible to determine approximately how much battery each one needed on average for each sampling interval. Let´s firstly remind the tested sampling intervals, 5, 10, 30, and 60 seconds. Only these intervals were tested and used for the evaluation of this work. Many others could be used but these were the ones chosen. The ideal would be to test all possible sampling times but that requires a lot of testing time. It is important to notice that the writings performed on the log file for evaluation purposes were all made with the same interval. They all were made with a 5 seconds interval. This value was defined because it is the greatest common divisor of them all.

This measure was taken because when the sampling interval is 5, the log registers everything every 5 seconds, writing on it. That spends battery that is not directly related with the application once this log only exists for testing purposes, it does not need to exist during the normal execution of the app. It must be considered that the continuous act of writing to a log file may be more expensive in terms of battery than the readings themselves. Although that is something really hard to test once without logs we would not be able to compare the number of readings performed during a 1% battery drop. So it is necessary to assure that the battery spent on writing to the log is constant no matter which sampling is tested.

This implies the only thing different between algorithms and samplings are only the algorithms power consumption that may need more or less battery to gather the sensors data, determine the activity and the samplings rate. Nothing else apart from those factors should be different between the two solutions. Only that way is possible to make the most accurate comparison between them and between each sampling interval for each algorithm. For that reason, it was decided to write something in the log every 5 seconds. Even when the sampling is not 5 seconds. In those cases, when there is no real need to write it is just wrote the word nothing with the timestamps. For instance, in a 30 seconds sampling interval, the log has a reading line and 5 consecutive lines with the word nothing, repeating this sequence over and over until the trip or the application is stopped. To make it easier to understand that it is provided Fig. 5.2. In there is displayed an example of an output of the GAR solution for the 30 seconds sampling interval. It is possible to see the values of each of the 8 activities ordered by id in an array. The cycling value has the id=1, what that means is that it is displayed in the second position. That is due to the semantics of the array. It starts counting on zero and not on one so the id is always the position minus

```
10, 10, 10, 10, 40, 0, 10, 10, 16:44:33, 83%
nothing 16:44:33, 16:44:38, 83%
nothing 16:44:33, 16:44:43, 83%
nothing 16:44:33, 16:44:48, 83%
nothing 16:44:33, 16:44:53, 83%
nothing 16:44:33, 16:44:58, 83%
10, 10, 10, 10, 40, 0, 10, 10, 16:45:03, 83%
nothing 16:44:33, 16:45:08, 83%
nothing 16:44:33, 16:45:13, 83%
nothing 16:44:33, 16:45:19, 83%
nothing 16:44:33, 16:45:24, 83%
nothing 16:44:33, 16:45:29, 83%
6, 28, 28, 30, 2, 0, 28, 2, 16:45:34, 83%
```

Figure 5.2: GAR 30 seconds output example

one. After the probabilities, it is displayed the time of writing in the log. This was to assure the readings and writings are being performed every 5 seconds.

Finally, the battery percentage to help us with this battery testing. To notice that when there are no readings and is written the word nothing on the file there are two timestamps. There the first represents the time of the creation of the service and the second the time of the writing. This was made due to the challenges that occurred during implementation as explained in Section 4.4. It is basically because the services were being killed by the Android system and were not launched immediately. And most of the time it took around 5 minutes or more to launch a new service to gather readings. This caused to appear some huge gaps in the log file where the applications were not gathering samples. So this was added to control if the service was constantly running and if it was the same service or a new one launched after the previous stopped.

	5	**10**	**30**	**60**
OP7	341	390	446	523
OPX	272	295	342	401
REDMI4	309	356	438	504

Table 5.11: GAR number of readings
First line has the sampling intervals
Other lines display the number of readings

	5	**10**	**30**	**60**
OP7	374	395	461	553
OPX	239	270	304	341
REDMI4	320	381	447	532

Table 5.12: RF number of readings
First line has the sampling intervals
Other lines display the number of readings

Table 5.11 and Table 5.12 display the number of readings performed while the battery percentage dropped 1%. That is achieved thanks to the log file. By analyzing it is possible to get the number of writes, associated with a specific reading, made during the same battery percentage level. The battery percentage level is associated with each write so it is possible to know the battery level in the log. This

testing does not counts/considers the final algorithm. It only considers and tests the battery spent during the trip recording, without the determination of the activity, just the sensors data gathering.

Here, by comparing both tables, is possible to immediately detect that the RF algorithm can perform more readings for the same amount of battery consumption. This shows that the battery used during the data gathering, i.e., read from the sensors, store and write those values in the log file is less for RF than GAR. The possible explanation for that may be the fact that the RF simply collects the raw data during the trip recording. On the other hand, the GAR collects that data from some unknown sensors but treats them, calculating their selected features and passing them through the algorithm returning each activity's probability. Due to this, the GAR solution is always performing more work for each reading it does and it seems that has an impact on the battery consumption. That occurs for all devices except for the OPX. For this device, all the values are lower than for the same sampling intervals on GAR. This one is the oldest and the less powerful in terms of computational capacity. Maybe for that reason, it may struggle more with the constant sensors readings, especially the GPS, which as seen before is one of the most power-demanding sensors. For the other two devices, the OnePlus 7 (OP7) and the Xiaomi Redmi 4 (REDMI4) the RF results were better.

For both solutions, the number of readings increases with the increase of the sampling interval. That was expected once the bigger the sampling interval the fewer the number of calls to the sensors. And fewer calls to the sensors means fewer readings performed in the same time interval. Which results in a lower battery consumption caused by the sensors. However, the number of readings is not directly proportional to the sampling intervals. For instance, the sampling interval of 60 seconds only performs one reading each minute while the 30 seconds sampling interval performs 2. That is two times more the number of readings in the same amount of time which should mean two times more battery spent. So the time to drop 1% of the battery should be two times faster in the smaller sampling interval, i.e., it should perform half the readings of the bigger sampling interval. That does not happen because the battery spent is not that linear but also because even when there are not being performed readings the apps are writhing on the file every 5 seconds. Those empty writings spend battery and have a direct influence on the final result. So that is why the number of readings between two different samplings does not vary as much as expected. Maybe without these writings, the difference between them would increase and be closer to the direct proportion. To notice that these writings are not necessary for the application's normal performance but are for testing purposes so that is the reason they have been used and kept here as well.

5.5 Summary

In sum, it was possible to observe that there was not a solution that outperformed the other in all three tested categories. The RF performed better in terms of accuracy. It presented better results for both device placements and tested activities. The differences presented between the two solutions are considerable, especially for the up/downstairs activity testing. Here the RF presented way fewer false positives. The difference is around 5 times less than the GAR. When it comes to the cycling detection the RF presented a smaller error. Looking separately at the placements, the RF improved on average 8,7% for the front pocket and 10% for the backpack. That is almost a 50% error reduction for both the front pocket and the backpack positions, resulting in an overall 50% error reduction when using RF instead of GAR. This solution also presented better results for the battery consumption test. It was able to perform more readings with the same amount of battery spent. This means that each reading is less battery expensive in the RF than the GAR. Although here the differences were not as big as for the accuracy. The results were more leveled in this testing. For the response time, the GAR presented better results. In this test, the differences are noticeable once the GAR response time is around 10 times faster than the RF. This is the test with bigger differences in results for both algorithms. That, as mentioned before, could be related to the fact the RF performs all the calculations in the end while the GAR performs it during the sampling time, only making an average of those cycling results in the end.

6

Conclusion

Contents

This chapter discusses the conclusions taken from this work, after all the research and development process. It describes as well some possible and interesting future work that could be done based on this work.

6.1 Conclusions

The increasing concern with the global environment due to a continuous increase of the global average temperature along with the decreasing of global natural resources there caused the world to take some actions. Actions aiming to improve those aspects of the environment. One of the most discussed issues that are helping to increase the global temperature is the CO2 gases. Gases those that are majority expelled by factories but also cars. So the reduction of the number of cars circulating daily around the world is a good measure to reduce those emissions and consequently decrease the temperature rising. This along with some health factors related to increasing obesity levels created an opportunity to remove cars from the roads and making people practicing physical exercise. There are many applications with that purpose, being Biklio the one that is interesting for this work. It offers some gifts/discounts to encourage people to replace the car or bus with a bicycle. This app detects if the user is cycling for that.

This work goes through several aspects of the necessary things to have a detection application working and ready. As shown before there are already many possible solutions for transportation mode detection and many different ways to implement it. There are solutions based only on the mobile phones of the users. Others are focused on external devices, either composed of a single sensor or a combination of them. Others that are a hybrid solution, i.e., a solution that relies on both users phone and external device. There are three main decisions to make in this type of works, which sensors to use, which features, and which algorithm. It was shown that there are several sensors capable of determining an activity. Also, it was shown that multiple features can be selected and extracted based on the selected sensors. Although for each sensor the amount of features is very high and they are not always used the same features in different works with the same sensors. Finally, the algorithm decision is equally hard because there are so many already applied for this type of application and presenting good results. Although it is necessary to understand which fits better for each case, having its advantages and disadvantages into consideration. The combination of these three decisions is almost infinite due to multiple options on each of the decisions.

From all the possible solutions existing it was decided to choose a ML algorithm to attempt to replace the GAR. The decision was to use the RF with the accelerometer and the GPS. The GAR and RF applications were implemented and tested to compare them and evaluate their performance in three main terms, accuracy, response time, and battery consumption. The RF presented better results for accuracy and battery consumption while the GAR presented better results for response time. During

the testing phase, it was possible to assess some of the expected outcomes. I.e., most of the time it was really difficult to determine/understand the reasons why the results exhibited by the GAR once the programmer does not know how it works and what sensors it uses. While on the other hand, the RF, is clear and transparent to the programmer, knowing all the used sensors, features, and algorithms. This made it easier to understand and explain the results obtained during the evaluation phase. However, this solution has its disadvantages. The main one is the necessary work to implement it. The programmer must test the sensors to see which are better. Test the features to understand if they are necessary and useful. Create the algorithm and adjust all the necessary values so it performs as it should. Gather readings and train the algorithm so it can build a successful model. Only then it can start determining the activity. The GAR in its turn only needs to be called and displays the results.

Overall, the proposed solution for this work, RF, presented better results. It outperformed considerably the GAR in the accuracy test, returning higher accuracy levels for both device placements. The number of false positives returned was also lower than the GAR. It also performed better for the battery test, being able to perform more readings spending the same amount of battery as the GAR. And although it had taken more time to return the result to the user, its response time is still nothing once it never reaches the one-second mark. This amount of time is basically nothing to the users. It is not an amount of time that requires the user to wait for the result. It is almost instantaneously displayed after stopped tracking and reached the destination. Both solutions presented fast response times but it is necessary to remember that this GAR solution had a simple average algorithm on top of it instead of the much more complex one discussed in Section 3.2. With that algorithm applied instead of this one the battery consumption but mostly the response time results would suffer from it and probably increase their values.

6.2 Future Work

This work presents several things that could be done as future work. For starters, the RF application can be tested in multiple different ways. It was only tested for cycling and stairs activities. The other activities can produce a high number of false positives and cause the accuracy of the app to drop.

Also, the values used on the features could be modified to see if it is possible to achieve higher accuracy. By changing slightly the windows of accepted values in the RF algorithm the accuracy may improve. There are several combinations and not all have been tested here due to time limitations.

Nowadays, there are multiple sensors available on smartphones that could be somehow used to determine the type of activities described in this work. So it would be interesting to see how this solution behaves with different sensors combination. In particular with the accelerometer, magnetometer, and gyroscope combination. This solution seems promising and presented interesting results in [24],

although it uses a different algorithm. It would be nice to see if those sensors could improve this application using the RF. The most interesting part here is that the possibility of replacing the GPS with the magnetometer, and gyroscope. This modification could, theoretically, mean a less battery hungry application once the GPS is one of the sensors that use most battery and the combination of the individual battery consumption of those two sensors is lower than the GPS alone.

There is equally the option of adding new features or removal of used features. The addition of a single feature can have a significant impact on the result of the application. The right features for this particular case must be tested. We only have tested some but there are still many that have not been tested and can help to improve the accuracy of the app or remove/decrease the number of false positives.

This is only for local applications. In case an app with an external sensor/device presents better results without compromising the rest of the work, i.e., without increasing significantly the battery consumption or response time due to the communications that need to be established with the external sensor/device.

Bibliography

[1] S. Saeedi and N. El-Sheimy, "Activity recognition using fusion of low-cost sensors on a smartphone for mobile navigation application," *Micromachines*, vol. 6, pp. 1100–1134, 2015.

[2] M.-C. Yu, T. Mengchieh, T. Yu, S.-C. W. HTC, T. D. Wang, htccom Chih-Jen Lin, E. Y. C. HTC, and T. Edward, "Big data small footprint: The design of a low-power classifier for detecting transportation modes," 2014.

[3] Air pollution - our world in data. Accessed 25-November-2019. [Online]. Available: https://ourworldindata.org/air-pollution

[4] H. Ritchie and M. Roser, "Air pollution," *Our World in Data*, 4 2017, accessed 25-November-2019. [Online]. Available: https://ourworldindata.org/air-pollution

[5] Causes, effects and impressive solutions to air pollution - conserve energy future. Accessed 25-November-2019. [Online]. Available: https://www.conserve-energy-future.com/causes-effects-solutions-of-air-pollution.php

[6] H. M. Mir, K. Behrang, M. T. Isaai, and P. Nejat, "The impact of outcome framing and psychological distance of air pollution consequences on transportation mode choice," *Transportation Research Part D: Transport and Environment*, vol. 46, pp. 328–338, 7 2016.

[7] Where in europe can i drive my diesel car? Accessed 25-November-2019. [Online]. Available: https://dieselinformation.aecc.eu/where-in-europe-can-i-drive-my-diesel-car/

[8] "Activityrecognitionclient — google play services — google developers," accessed 9-May-2021. [Online]. Available: https://developers.google.com/android/reference/com/google/android/gms/location/ActivityRecognitionClient

[9] B. Ictech, "Smartphones and face-to-face interaction: Digital cross-talk during encounters in everyday life," *Symbolic Interaction*, vol. 42, pp. 27–45, 2 2019, accessed 25-November-2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/symb.406https://onlinelibrary.wiley.com/doi/abs/10.1002/symb.406https://onlinelibrary.wiley.com/doi/10.1002/symb.406

[10] X. Su, H. Tong, and P. Ji, “Activity recognition with smartphone sensors,” 2014.

[11] Number of connected devices per person — statista. Accessed 6-December-2019. [Online]. Available: https://www.statista.com/statistics/678739/forecast-on-connected-devices-per-person/

[12] Biklio. Accessed 9-March-2020. [Online]. Available: https://www.biklio.com/

[13] A. Jylhä, P. Nurmi, M. Sirén, S. Hemminki, and G. Jacucci, “Matkahupi: A persuasive mobile application for sustainable mobility,” 2013, pp. 227–230.

[14] Registar a atividade diária com o apple watch - suporte apple. Accessed 17-May-2020. [Online]. Available: https://support.apple.com/pt-pt/guide/watch/apd3bf6d85a6/watchos

[15] Indicações, estado do trânsito e boleias com o waze. Accessed 17-May-2020. [Online]. Available: https://www.waze.com/

[16] Activity recognition api — google developers. Accessed 1-December-2019. [Online]. Available: https://developers.google.com/location-context/activity-recognition

[17] S. Hemminki, P. Nurmi, and S. Tarkoma, “Accelerometer-based transportation mode detection on smartphones.” Association for Computing Machinery, 2013.

[18] K. Murao and T. Terada, “A recognition method for combined activities with accelerometers.” Association for Computing Machinery, Inc, 2014, pp. 787–796.

[19] S. Ranasinghe, F. A. MacHot, and H. C. Mayr, “A review on applications of activity recognition systems with regard to performance and evaluation,” 8 2016.

[20] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, “Deep learning for sensor-based activity recognition: A survey,” *Pattern Recognition Letters*, vol. 119, pp. 3–11, 3 2019.

[21] T. Sohn, A. Varshavsky, A. LaMarca, M. Y. Chen, T. Choudhury, I. Smith, S. Consolvo, J. Hightower, W. G. Griswold, and E. D. Lara, “Mobility detection using everyday gsm traces,” vol. 4206 LNCS. Springer Verlag, 2006, pp. 212–224.

[22] W. Wang, A. X. Liu, M. Shahzad, K. Ling, and S. Lu, “Understanding and modeling of wifi signal based human activity recognition,” vol. 2015-September. Association for Computing Machinery, 9 2015, pp. 65–76.

[23] V. C. Coroamă, C. Türk, and F. Mattern, “Exploring the usefulness of bluetooth and wifi proximity for transportation mode recognition.” Association for Computing Machinery, Inc, 9 2019, pp. 37–40.

[24] S. H. Fang, H. H. Liao, Y. X. Fei, K. H. Chen, J. W. Huang, Y. D. Lu, and Y. Tsao, “Transportation modes classification using sensors on smartphones,” *Sensors (Switzerland)*, vol. 16, 8 2016.

[25] S. Liu, R. X. Gao, L. Mo, and P. S. Freedson, “Wearable sensing for physical activity measurement: Design and performance evaluation,” vol. 46. IFAC Secretariat, 2013, pp. 53–60.

[26] J. Liono, Z. S. Abdallah, A. K. Qin, and F. D. Salim, “Inferring transportation mode and human activity from mobile sensing in daily life.” Association for Computing Machinery, 11 2018, pp. 342–351.

[27] P. Ferreira, C. Zavgorodnii, and L. Veiga, “edgetrans - edge transport mode detection,” *Pervasive and Mobile Computing*, vol. 69, p. 101268, 11 2020.

[28] T. Choudhury, S. Consolvo, B. Harrison, J. Hightower, A. LaMarca, L. LeGrand, A. Rahimi, A. Rea, G. Borriello, B. Hemingway, D. Haehnel, P. Klasnja, K. Koscher, J. A. Landay, J. Lester, and D. Wyatt, “The mobile sensing platform: An embedded activity recognition system,” *IEEE Pervasive Computing*, 2008.

[29] J. Liang, Q. Zhu, M. Zhu, M. Li, X. Li, J. Wang, S. You, and Y. Zhang, “An enhanced transportation mode detection method based on gps data,” vol. 727. Springer Verlag, 2017, pp. 605–620.

[30] L. Breiman, “Random forests,” pp. 5–32, 2001.

[31] Y. Freund and R. E. Schapire, “Experiments with a new boosting algorithm,” 1996. [Online]. Available: http://www.research.att.com/

[32] How do machine learning algorithms differ from traditional algorithms? Accessed 7-June-2020. [Online]. Available: https://analyticsindiamag.com/how-do-machine-learning-algorithms-differ-from-traditional-algorithms/

[33] I. Anderson and H. Muller, “Practical activity recognition using gsm data *.”

[34] D. Montoya, S. Abiteboul, and P. Senellart, “Hup-me: Inferring and reconciling a timeline of user activity from rich smartphone data,” vol. 03-06-November-2015. Association for Computing Machinery, 11 2015.

[35] V. Mihajlovic and M. Petkovic, “Dynamic bayesian networks- a state of the art.”

[36] L. Stenneth, O. Wolfson, P. S. Yu, and B. Xu, *Transportation Mode Detection using Mobile Phones and GIS Information*.

[37] C. Zhou, H. Jia, J. Gao, L. Yang, Y. Feng, and G. Tian, “Travel mode detection method based on big smartphone global positioning system tracking data,” *Advances in Mechanical Engineering*, vol. 9, 6 2017.

[38] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, "Using mobile phones to determine transportation modes," *ACM Transactions on Sensor Networks*, vol. 6, 2 2010.

[39] D. Shin, D. Aliaga, B. Tunçer, S. M. Arisona, S. Kim, D. Zünd, and G. Schmitt, "Urban sensing: Using smartphones for transportation mode classification," pp. 76–86, 2015.

[40] S. H. Fang, Y. X. Fei, Z. Xu, and Y. Tsao, "Learning transportation modes from smartphone sensors based on deep neural network," *IEEE Sensors Journal*, vol. 17, pp. 6111–6118, 9 2017.

[41] Deep neural networks - kdnuggets. Accessed 7-June-2020. [Online]. Available: https://www.kdnuggets.com/2020/02/deep-neural-networks.html

[42] J. Pärkkä, M. Ermes, P. Korpipää, J. Mäntyjärvi, J. Peltola, and I. Korhonen, "Activity classification using realistic data from wearable sensors," *IEEE Transactions on Information Technology in Biomedicine*, vol. 10, pp. 119–128, 1 2006.

[43] Y. Zheng, Y. Chen, Q. Li, X. Xie, and W. Y. Ma, "Understanding transportation modes based on gps data for web applications," *ACM Transactions on the Web*, vol. 4, 1 2010.

[44] Weka 3 - data mining with open source machine learning software in java. Accessed 10-May-2020. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/

[45] Java data mining package — machine learning and big data analytics. Accessed 10-May-2020. [Online]. Available: https://jdmp.org/

[46] Machine learning library (mllib) programming guide - spark 1.2.0 documentation. Accessed 10-May-2020. [Online]. Available: https://spark.apache.org/docs/1.2.0/mllib-guide.html

[47] "Advantages of java," accessed 14-June-2021. [Online]. Available: https://www.ibm.com/docs/en/aix/7.1?topic=monitoring-advantages-java

[48] "Services overview — android developers," accessed 9-May-2021. [Online]. Available: https://developer.android.com/guide/components/services

[49] "Android os version market share over time — appbrain," accessed 14-June-2021. [Online]. Available: https://www.appbrain.com/stats/top-android-sdk-versions

[50] "Bu-802: What causes capacity loss? - battery university," accessed 14-June-2021. [Online]. Available: https://batteryuniversity.com/article/bu-802-what-causes-capacity-loss

[51] "How to properly charge a phone battery," accessed 14-June-2021. [Online]. Available: https://www.techadvisor.com/how-to/mobile-phone/charge-phone-properly-3619623/

A

Code of Project

In this appendix is displayed some code that was referenced in the text but is too specific to be explained and presented there.

The following code represents the necessary code for the android.permission.REQUEST_IGNORE_BATTERY_OPTIM implementation described in Section 4.4. It shows what is the necessary code to implement, for both Java code in the activity and the XML file. The permissions should be declared on the XML file. The Java code should be asked to the user to remove battery optimizations for the device. This code sends him directly to the app battery definitions to do that. This XML file also shows the necessary permissions for the GAR with the ACTIVITY_RECOGNITION permission and the permission to write to the device's storage with WRITE_EXTERNAL_STORAGE. The last one asked because of the log file.

Listing A.1: Example of a XML file with GAR and request to ignore battery optimization.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="packageName">
```

```
4
5       <uses-permission android:name="com.google.android.gms.permission.ACTIVITY_RECOGNITION" />
6
7       <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
8
9       <uses-permission
            android:name="android.permission.android.permission.REQUEST_IGNORE_BATTERY_OPTIMIZATIONS"/>
10
11      <application
12          .
13          .
14          .
15      </application>
16
17  </manifest>
```

Next, the necessary Java code that needs to be implemented in the main activity, i.e., the application initial activity for both writing and to ignore battery optimization.

Listing A.2: `Java code`

```
18  public class MainActivity
19  {
20
21      @Override
22      public void onCreate(Bundle savedInstanceState) {
23          super.onCreate(savedInstanceState);
24          setContentView(R.layout.activity_main);
25          mContext = this;
26
27          //ask for writing permissions
28          checkPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE);
29
30          //asks to disable battery optimization
31          Sentiance.getInstance(mContext).disableBatteryOptimization();
32      }
33  }
```

The code of both solutions of this work is open source and is available on GitHub in the following links:

- The GAR solution is available at `git@github.com:brunoaosantos/TestApp.git`.
- The RF solution is available at `git@github.com:brunoaosantos/DetectBiklioML-master.git`.