DetectBiklio - detect bicycle usage with an Android smartphone app

Bruno Santos
bruno.o.santos@tecnico.ulisboa.pt
Instituto Superior Técnico
Lisbon, Lisbon, Portugal

Figure 1. Biklio logo

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.

Instituto Superior Técnico, July 25, 2021, Lisbon

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

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

ACM Reference Format:
Bruno Santos. 2021. DetectBiklio - detect bicycle usage with an Android smartphone app. In *Proceedings of July 25, 2021 (Instituto Superior Técnico).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

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 [4, 24]. There are numerous causes for the increase of such pollution being one of them the motorized vehicles [6, 23]. 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 [13] 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 [2]. 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 [19]. In fact, smartphones are becoming ubiquitous in nowadays society [26]. Everywhere we go, people are using their smartphones. That is noticeable looking at the continuous increase of devices per person nowadays [10]. 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 (CO2) emission levels, and the increasing obesity numbers, created a pretty interesting opportunity for the development of a new type of apps.

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 [5], (the one we are most interested in) or MatkaHupi [20]; 2) activity tracking apps for healthcare such as Activity [11]; 3) finally, activity-driven crowdsourcing apps such as Waze [7]. All these apps have in common that they are all based on activity recognition. Some of the available sensors on the smartphone [26] 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. For that it uses the Google Activity Recognition (GAR) Application Programming Interface(API) [1].

1.1 Objectives

The main objective of this work is to improve the existing cycling detection algorithm of the Biklio app [5]. 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. An android version is implemented to understand if the app algorithm improved the current solution.

2 Related Work

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. In this work the activity is associated with the transportation mode. The aim of this work is an application that determines the users transportation mode, cycling more precisely. This type of activity is the most interesting for this particular case, once Biklio aims to detect cycling activities.

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 GSM (e.g. Sohn [25]), WIFI (e.g. Wang [27]), Bluetooth (e.g. Coroamă [15]), or a combination of them. Motion detection sensors, such as accelerometers or gyroscope (e.g. Fang [16])

can also be used to determine transportation modes. Those solutions can be external (e.g. Liu [22]) or internal (e.g. Hemminki [18]). 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 [21] and Ferreira [17]) algorithms to determine the transportation mode. Some others use existing APIs like GAR [1] that can be used to accomplish the goal.

Activity recognition systems typically have three main components [14]: 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.

After analyzed several similar works two different approaches seemed interesting. One was the Google Activity Recognition (GAR) and the other was the Random Forest (RF). The GAR was a simpler option once it is a pre-built Application Programming Interface (API) that performs all the detection work. The three main components are integrated into the API. The only thing that is necessary to use it is to call the API. The RF is harder once it is necessary to choose and implement all the three components necessary for an activity recognition system, sensors, features, and algorithm.

3 Architecture/Algorithm

There were implemented two different solutions that were later compared.

The first was the Google Activity Recognition (GAR) Application Programming Interface (API). As mentioned in Section 2 this solution encapsulates the three main components of an activity recognition system. This results in a simpler solution design (see Fig. 2), once those components do not need to be implemented. They only need to be called and the API performs all the work for the user returning only the final result.

This solution's algorithm is something the programmer does not has access to, just like the rest of the information, sensors, and feature selection namely. What this means is the programmer does not have a lot of control and knowledge over the solution. It will always be dependent on the performance of the API. If the API fails the application will fail and the work necessary to understand/fix the problem is bigger once the programmer does not know what sensors,

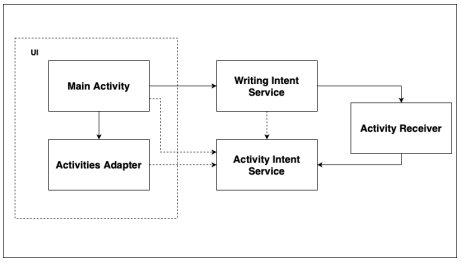

Figure 2. GAR architecture
Dotted arrows mean content access
Full arrows mean the creation of the component

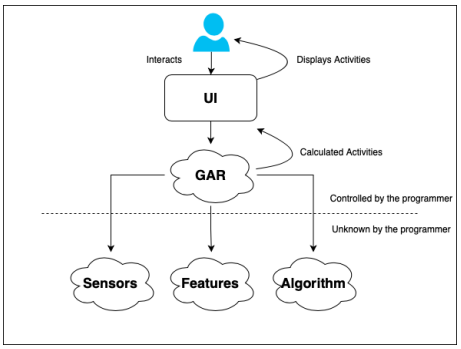

Figure 3. GAR representation
Cloud represents unknown info

features, and algorithms are used. That is presented in Fig. 3, where the cloud symbols represent the unknown information. Shows the user/application making a call to the GAR and the GAR handles the call by calling the selected sensors, calculating the selected features, and run them through the selected algorithm.

This algorithm is continuously running. 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).

The probabilities vary between 0 and 100%. Only the activities with a current probability higher than 0 are returned. 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.

The other implemented solution was the RF, a Machine Learning (ML) algorithm. This application's architecture design is much harder than the GAR. That is because this solution does not have the three components (sensors, features,

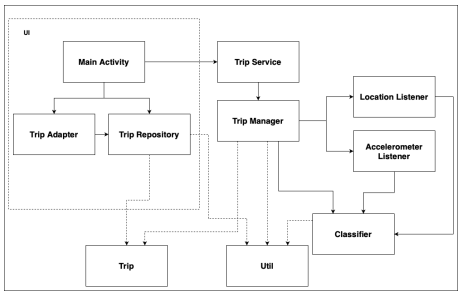

Figure 4. RF architecture
Dotted arrows mean content access
Full arrows mean the creation of the component

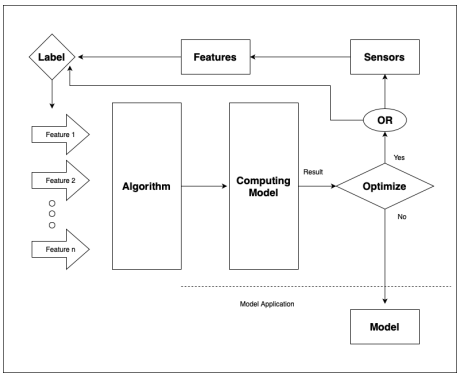

Figure 5. RF Trainig phase

and algorithm) encapsulated. This requires the programmer to implement them resulting in a higher complexity level and a higher number of components needed. The Fig. 4 supports that same affirmation. When compared to the GAR's architecture (see Fig. 2) the number of components present here is considerably higher.

Just like the architecture, the algorithm of this solution is also harder when compared to the GAR. This is mainly because the algorithm needs to be chosen, coded/implemented, trained, and only then used. Each of these steps needs some work. All essential to achieve the main goal of this work which is to determine accurately the cycling activity.

Here the chosen algorithm was the RF and after being implemented it was needed to be tested. In particular, it was used, as a base model for this solution, the Woorti [17] application. The testing phase is very important because it is the foundation of the application's success. This is since from here is going to result in the model that is going to be used for the activity prediction when the user runs the app. 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. 5 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. 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. 6 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. 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 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).

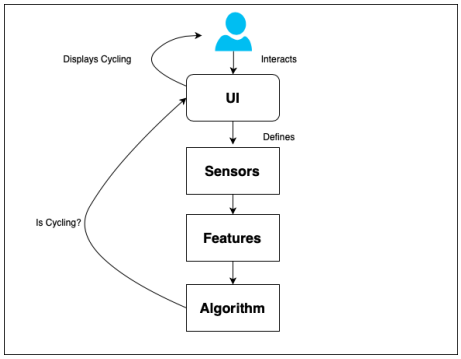

Figure 6. RF prediction phase

4 Implementation

As mentioned in Section 1 as the fourth and last requirement, Android was chosen as the target Operative System (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 most important 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 [12], JDMP [8], and MLib (Spark) [9]. 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 [3]. 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.

For each solution was built an app. For the GAR that app displayed all the possible transportation modes detected by it 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 Section 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.

All the 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. It presents the 8 types of activities detected by the GAR.

To present those values it is necessary to call the GAR API. The app performs the sensor's readings. 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.

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 [2]. 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.

The RF application requires a different implementation. 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. 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.

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.

5 Evaluation

Both solutions were tested under the same conditions, as far as possible. There were used 3 different devices, the Oneplus 7, Oneplus X, and Xiaomi Redmi Note 4. Each of these devices has a different Android version running, sensors versions, and computational power. This way both solutions can be tested for a wider type of devices, instead of only one specific type. Each approach was tested for two different positions. One was in the front pocket of the pants and the other was in a backpack. 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.

To avoid/minimize the differences between cycling trips, the route performed during the testing phase was always the same. That is 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.

Each solution was tested in three main fields, accuracy, response time, and battery. These are the first three requirements discussed in Section 1.

5.1 Accuracy

For each solution (GAR and RF), the accuracy-test was divided into two main components, device placement, and activity. Each algorithm was tested and their results were discussed.

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 1 and Table 3 show the results obtained from those tests for the front pocket for cycling and stairs respectively, while Table 2 and Table 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 1 and Table 2 (tested while cycling) the higher the values the better the accuracy, in Table 3 and Table 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 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 2. GAR cycling accuracy backpack
First line has the sampling intervals
Other lines display the accuracy (%)

The cycling results (Table 1 and Table 2) shows 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.

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

Table 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 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. They are also very similar for both device placements.

Moving on to the RF the exact same tables were constructed. Table 5 and Table 7 show the results obtained from the tests for the front pocket for cycling and stairs respectively, while Table 6 and Table 8 for backpack.

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

Table 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 6. RF cycling accuracy backpack
First line has the sampling intervals
Other lines display the accuracy (%)

In Table 5 and Table 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.

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

Table 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 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 7 and Table 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.

5.2 Response Time

The 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.

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

Table 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 10. RF average time
First line has the sampling intervals
Other lines display the response time (ms)

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.

5.3 Battery

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.

Both these solutions write in a log file every 5 seconds. Those writes are performed only for evaluation purposes. They are not necessary for both application's normal execution. This value was defined because it is the greatest common divisor of all the samplings available, 5, 10, 30, and 60 seconds. These continuous writes take some computational power which increases battery consumption. So the results here are influenced by the battery spent on writing. However, with this is possible to guarantee that the impact of the writings is the same for all solutions and samplings. The only difference is if the writing is empty or has a correspondent reading associated.

Table 11 and Table 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.

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. 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 RF results were better.

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

Table 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 12. RF number of readings
First line has the sampling intervals
Other lines display the number of readings

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.

6 Conclusion

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 Carbon Dioxide (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 Machine Learning (ML) algorithm to attempt to replace the Google Activity Recognition (GAR). The decision was to use the Random Forest (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.

7 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 [16], 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.

References

[1] [n.d.]. Activity Recognition API | Google Developers. https://developers.google.com/location-context/activity-recognition Accessed 1-December-2019.

[2] [n.d.]. ActivityRecognitionClient | Google Play services | Google Developers. https://developers.google.com/android/reference/com/

google/android/gms/location/ActivityRecognitionClient Accessed 9-May-2021.

[3] [n.d.]. Advantages of Java. https://www.ibm.com/docs/en/aix/7.1?topic=monitoring-advantages-java Accessed 14-June-2021.

[4] [n.d.]. Air Pollution - Our World in Data. https://ourworldindata.org/air-pollution Accessed 25-November-2019.

[5] [n.d.]. Biklio. https://www.biklio.com/ Accessed 9-March-2020.

[6] [n.d.]. Causes, Effects and Impressive Solutions to Air Pollution - Conserve Energy Future. https://www.conserve-energy-future.com/causes-effects-solutions-of-air-pollution.php Accessed 25-November-2019.

[7] [n.d.]. Indicações, estado do trânsito e boleias com o Waze. https://www.waze.com/ Accessed 17-May-2020.

[8] [n.d.]. Java Data Mining Package | Machine Learning and Big Data Analytics. https://jdmp.org/ Accessed 10-May-2020.

[9] [n.d.]. Machine Learning Library (MLlib) Programming Guide - Spark 1.2.0 Documentation. https://spark.apache.org/docs/1.2.0/mllib-guide.html Accessed 10-May-2020.

[10] [n.d.]. Number of connected devices per person | Statista. https://www.statista.com/statistics/678739/forecast-on-connected-devices-per-person/ Accessed 6-December-2019.

[11] [n.d.]. Registar a atividade diária com o Apple Watch - Suporte Apple. https://support.apple.com/pt-pt/guide/watch/apd3bf6d85a6/watchos Accessed 17-May-2020.

[12] [n.d.]. Weka 3 - Data Mining with Open Source Machine Learning Software in Java. https://www.cs.waikato.ac.nz/ml/weka/ Accessed 10-May-2020.

[13] [n.d.]. Where in Europe can I drive my diesel car? https://dieselinformation.aecc.eu/where-in-europe-can-i-drive-my-diesel-car/ Accessed 25-November-2019.

[14] Tanzeem Choudhury, Sunny Consolvo, Beverly Harrison, Jeffrey Hightower, Anthony LaMarca, Louis LeGrand, Ali Rahimi, Adam Rea, Gaetano Borriello, Bruce Hemingway, Dirk Haehnel, Predrag Klasnja, Karl Koscher, James A. Landay, Jonathan Lester, and Danny Wyatt. 2008. The mobile sensing platform: An embedded Activity Recognition system. *IEEE Pervasive Computing* (2008).

[15] Vlad C. Coroamă, Can Türk, and Friedemann Mattern. 2019. Exploring the usefulness of bluetooth and WiFi proximity for transportation mode recognition. *UbiComp/ISWC 2019- - Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers*, 37–40. https://doi.org/10.1145/3341162.3343847

[16] Shih Hau Fang, Hao Hsiang Liao, Yu Xiang Fei, Kai Hsiang Chen, Jen Wei Huang, Yu Ding Lu, and Yu Tsao. 2016. Transportation modes classification using sensors on smartphones. *Sensors (Switzerland)* 16 (8 2016). Issue 8. https://doi.org/10.3390/s16081324

[17] Paulo Ferreira, Constantin Zavgorodnii, and Luís Veiga. 2020. edgeTrans - Edge transport mode detection. *Pervasive and Mobile Computing* 69 (11 2020), 101268. https://doi.org/10.1016/J.PMCJ.2020.101268

[18] Samuli Hemminki, Petteri Nurmi, and Sasu Tarkoma. 2013. Accelerometer-based transportation mode detection on smartphones. *SenSys 2013 - Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. https://doi.org/10.1145/2517351.2517367

[19] Brad Ictech. 2019. Smartphones and Face-to-Face Interaction: Digital Cross-Talk During Encounters in Everyday Life. *Symbolic Interaction* 42 (2 2019), 27–45. Issue 1. https://doi.org/10.1002/SYMB.406 Accessed 25-November-2019.

[20] Antti Jylhä, Petteri Nurmi, Miika Sirén, Samuli Hemminki, and Giulio Jacucci. 2013. Matkahupi: A persuasive mobile application for sustainable mobility. *UbiComp 2013 Adjunct - Adjunct Publication of the 2013 ACM Conference on Ubiquitous Computing*, 227–230. https://doi.org/10.1145/2494091.2494164

[21] Jonathan Liono, Zahraa S. Abdallah, A. K. Qin, and Flora D. Salim. 2018. Inferring transportation mode and human activity from mobile sensing in daily life. *ACM International Conference Proceeding Series*, 342–351. https://doi.org/10.1145/3286978.3287006

[22] Shaopeng Liu, Robert X. Gao, Lingfei Mo, and Patty S. Freedson. 2013. Wearable sensing for physical activity measurement: Design and performance evaluation. *IFAC Proceedings Volumes (IFAC-PapersOnline)* 46, 53–60. Issue 5. https://doi.org/10.3182/20130410-3-CN-2034.00073

[23] Hussein M. Mir, Koorosh Behrang, Mohammad T. Isaai, and Pegah Nejat. 2016. The impact of outcome framing and psychological distance of air pollution consequences on transportation mode choice. *Transportation Research Part D: Transport and Environment* 46 (7 2016), 328–338. https://doi.org/10.1016/j.trd.2016.04.012

[24] Hannah Ritchie and Max Roser. 2017. Air Pollution. *Our World in Data* (4 2017). https://ourworldindata.org/air-pollution Accessed 25-November-2019.

[25] Timothy Sohn, Alex Varshavsky, Anthony LaMarca, Mike Y. Chen, Tanzeem Choudhury, Ian Smith, Sunny Consolvo, Jeffrey Hightower, William G. Griswold, and Eyal De Lara. 2006. Mobility detection using everyday GSM traces. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 4206 LNCS, 212–224. https://doi.org/10.1007/11853565_13

[26] Xing Su, Hanghang Tong, and Ping Ji. 2014. Activity Recognition with Smartphone Sensors. Issue 3.

[27] Wei Wang, Alex X. Liu, Muhammad Shahzad, Kang Ling, and Sanglu Lu. 2015. Understanding and modeling of WiFi signal based human activity recognition. *Proceedings of the Annual International Conference on Mobile Computing and Networking, MOBICOM* 2015-September, 65–76. https://doi.org/10.1145/2789168.2790093