TÉCNICO LISBOA

DetectP2P

P2P Detection of Travel Mode

João Pedro Santana Calisto

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Paulo Jorge Pires Ferreira
Prof. Luís Manuel Antunes Veiga

Examination Committee

Chairperson: Prof. Daniel Jorge Viegas Gonçalves
Supervisor: Prof. Paulo Jorge Pires Ferreira
Member of the Committee: Prof. João Pedro Faria Mendonça Barreto

November 2019

Acknowledgments

Firstly, I would like to thank my family for their support through all these years and for the sacrifices that allowed me to finish my studies away from home. I would also like to thank my dissertation supervisors Prof. Paulo Ferreira and Prof. Luís Veiga for their insight and guiding. Lastly, I want to thank my girlfriend and all my friends for keeping me motivated and providing such incredible experiences.

Abstract

The detection of the users' travel mode is becoming more relevant due to the ubiquity of mobile devices and the applicability of this technology in multiple contexts. Many solutions can be found in the literature that aim at identifying the transport mode. However, some problems still exist due to the number of variables that negatively impact the system's accuracy, the device's power consumption, detection delays, etc. Therefore, in this thesis, we propose a new solution that combines a common machine learning technique with a P2P Network. This network allows the applications running in each device to exchange information and, consequently, improve the accuracy of the classifier. We believe that this solution provides higher confidence levels for each detection while maintaining a near real-time transport identification.

Keywords

Transport mode detection; Classification; Smartphone; P2P Network;

Resumo

A deteção do modo de transporte dos utilizadores tem sido um campo cada vez mais relevante devido à ubiquidade dos dispositivos móveis e da utilidade deste conhecimento em vários campos. Podemos encontrar várias soluções no âmbito de deteção de viagens. No entanto, ainda há muitos problemas relacionados com a exatidão do sistema, o consumo de bateria, a deteção em tempo-real, etc. Neste trabalho, é proposta uma nova solução que combina uma técnica de aprendizagem automática com uma rede de utilizadores a comunicar entre si. A aplicação, que corre no dispositivo de cada utilizador, troca informação com as outras instâncias da aplicação, a correr noutros dispositivos móveis (i.e., telemóvel), com o objectivo de melhorar a previsão local do classificador. Esta solução permite aumentar a precisão da deteção de viagens mantendo os requisitos de deteção em tempo real.

Palavras Chave

Deteção; Classificação; Modo de Transporte; Rede P2P;

Contents

List of Figures

List of Tables

1

Introduction

Contents

1.1 Motivation

Travelling is a frequent activity for the average human being. Many reasons for travelling can be found, either for recreation, work, socializing or any other personal motive. Mode of transportation is everyone's concern. Each person must balance variables such as time spent, distance travelled, comfort, energy consumed, greenhouse gas emissions and many more. The weight of each variable diverges between different people due to one's own preferences and lifestyle. These differences in human behaviour have been the object of scientific studies, namely project MoTiV (Mobility and Time Value) [1].

This work is developed in the scope of the MoTiV project, financed by Horizon 2020 [2]. The MoTiV project focuses on investigating the Value of Travel Time (VTT), introducing and validating a conceptual framework for the estimation of the VTT. Many researchers have been studying the VTT, defining it as the amount a person has to pay to go from one point to another from an economical perspective. MoTiV targets other aspects of VTT such as individual preferences, motivations and behaviours. A wider definition of VTT can be found when we consider the overall user activities, satisfaction and specific needs, which can be used to better understand travel decision making. Some people might prioritize more economical modes of transportation while others might choose alternatives that offer superior comfort.

In order to qualify and quantify the combinations of individual needs that lead to travellers' decisions, travel patterns must be observed. MoTiV collects mobility data from European users through a smartphone application [3]. Collected data is made available to the scientific community, allowing for further research on this topic to be continued. The collection process is facilitated by developing automated detection mechanisms, eliminating the need of having a user manually inserting information. Automated detection mechanisms must be efficient and accurate to ensure a trustworthy data set while requiring minimal resource usage. Analyzing and understanding the population behaviour is key to improve transport infrastructures resulting in a positive economic and environmental impact, along with social well-being.

1.2 Objectives

The goal in this work is to develop a solution that detects user's travel mode (i.e., travelling on foot, bicycle, train, car, bus) through a smartphone application (DetectP2P) that communicates, in real-time and without requiring internet access, with the surrounding instances of the application running in other devices. DetectP2P aims at using the communication between devices to offer more resistance against sensor noise and allowing devices to share mobility knowledge that is then used to improve future decisions.

Our approach explores one alternative to current solutions that either require a remote server to

execute the detection algorithm or directly execute it on the device itself (i.e., stand-alone), with all the disadvantages that we discuss in the related work section (see Chapter 2).

We describe and analyze multiple solutions related to the detection of the transport mode, including data collection methods for improved battery consumption, and the algorithms used to perform the detection. The knowledge acquired in the related work study is used and adapted to the solution developed in this work.

Our solution fulfils these fundamental requirements:

1. Each trip must be automatically detected, without requiring the user to manually indicate the start or end of a trip.
2. The mode of transportation must be correctly detected with adequate accuracy (at the minimum 85%, ideally over 90%), in order to produce the automatically collected results that are needed in the study of users' behaviour.
3. The application does not have internet access to maps, routes or transport itineraries.
4. The detection algorithm must output the optimal choice in less than 30 seconds, following the ending of a trip to assure the utility of the collected data (e.g., for traffic analysis).
5. The smartphone should have an expected battery life-time of at least 10 hours while running the application, to assure the complete collection of travel segments in the average day of an individual.

1.3 Main Challenges

There is no straightforward approach to detect the mode of transportation. Smartphone sensors (e.g., accelerometer, GPS, gyroscope) give us representations of the user movement including acceleration, speed, location, direction, etc. In a simplified approach, these readings could be compared against a model made of collected data in a test environment to identify the correct transport mode. However, several challenges arise as we describe below.

The sensor readings depend on multiple factors including variables such as the smartphone hardware (e.g., sensors with different sensibility), how the user transports the device, user-specific behaviour, possible interferences and many more. The readings from one user walking with the device in his pocket are different from the readings of that same user walking with the device in his hand, adding additional acceleration from his arm movement. Noisy data can have a significant impact on the accuracy of the model, particularly in solutions based on expert models (i.e., manually defined rules). Machine learning techniques are less sensitive to this problem, according to Ross & Kelleher [4]. Noise cancelling methods are useful to extract the actual movement features that are relevant for the transport mode detection and, consequently, improve the classification accuracy.

Additionally, every trip has unique characteristics (e.g., acceleration and velocity values). While it is possible to identify specific properties (e.g., maximum speed, acceleration range) that suggest the travel mode, the conditions are quite variable. Each class of transports contains multiple differences within its subclasses. For example, the class car includes an extensive variety of vehicles with different characteristics (e.g., acceleration rates, maximum speed, cornering speed, etc.). This concept is also applicable to other modes, with different characteristics for buses, trains and bicycles. In addition to vehicle properties, there are also external variables such as road types. Roads can have different properties, with more or fewer curves, speed limits, pavement, etc. Such variables impact the information perceived by the smartphone sensors and increase the difficulty of performing consistent classifications with such variable conditions.

Another problem is related to the device power consumption which is mainly affected by smartphone sensors [5]. Battery capacity is a limiting factor, particularly on data readings leading the applications to efficiently choose when and which sensors to use. Some solutions first use less power-hungry methods (e.g., accelerometer) to detect movement with low accuracy, changing to more accurate methods only when movement is detected (e.g., GPS), as we will see later. Not only the sensor types affect battery consumption but also the sampling frequency. The sensors' sampling rates can be adjusted to the application needs. High rates will have a significant impact on battery life but will offer better accuracy when compared with lower rates [6]. There must be a trade-off between accuracy and power consumption to ensure correct travel mode detection while maintaining the expected battery life-time.

The relation between the level of confidence (i.e., accuracy) obtained and the time required to perform the detection represents an additional challenge. Solutions that require a remote server (e.g., in the cloud) use more complex algorithms and offer a higher level of confidence when compared with a stand-alone solution due to the resource disparity (i.e., computational power). However, in the second approach, the application shows more responsiveness because there are no additional data transfer delays or connection-related problems [7].

1.4 Limitations of Existing Solutions

There are multiple solutions that focus on detecting travel mode [8]. Current solutions are divided between two main groups: stand-alone (see Section 2.1) and remote classification (see Section 2.2). Stand-alone solutions (i.e., executed in the smartphone) have limited resources to perform intensive computations, which impact the amount of identified transport modes with high accuracy. In approaches that require a remote server, the main limitation is the time it takes to perform the whole classification process. The use of a main server also translates to a possible bottleneck and scalability problems [9].

Solutions based in geodata knowledge[1] (see Section 2.2.1) fail to provide a wide coverage around the world due to the fact that some data (e.g., real-time bus location) is only available in certain regions [10]. This is also true for techniques that only require the GPS sensor, due to the lack of signal in certain locations (e.g., underground, dense tree zone, atmospheric effects, etc.). Additionally, the exclusive use of the GPS prevents the application from distinguishing the transport mode in certain conditions (e.g., intense traffic, limited speed areas) because the travelling speed is more limited by the external conditions than by the vehicle itself.

The poor management of sensors resulting in a significant reduction of the battery life represents an additional disadvantage in current solutions. Energy demanding applications are prone to be uninstalled by the user. Some solutions (e.g., [11], [12], [13], etc.) do not make any effort to reduce the power consumption of their system.

1.5 Proposed Solution

The main contribution of our work is the integration of the techniques found in the literature with information being exchanged among users sharing the same mode of transport. In this solution, the machine learning approach used in Woorti [14] is enhanced with mobility data being shared between users in a peer-to-peer environment. Such data includes the broadcast of transport mode predictions in real-time and previous trip validations confirmed by the users.

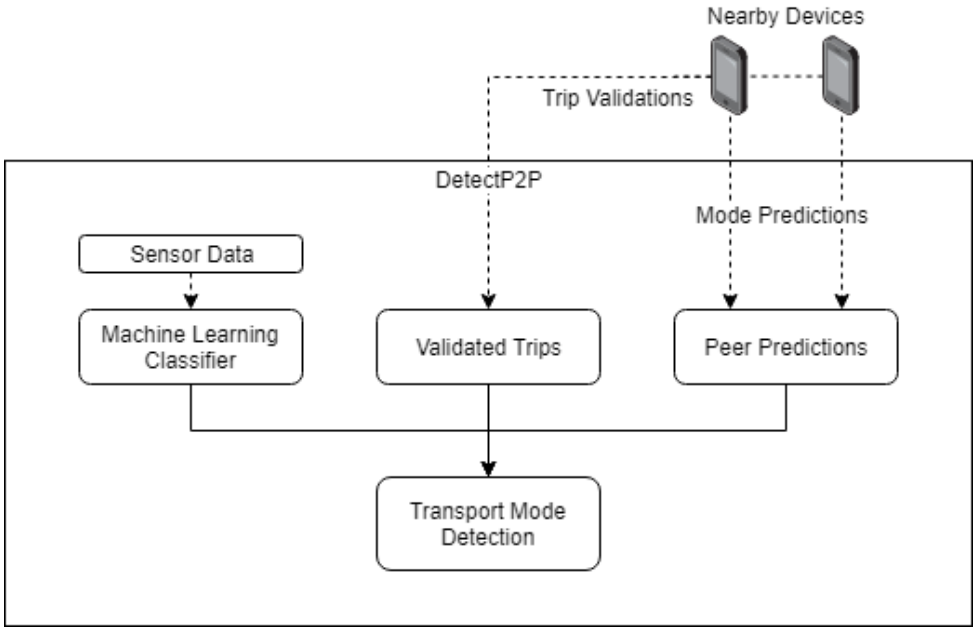

Figure 1.1: DetectP2P Overview

[1] Access to maps, routes and real-time location of public transports.

The application, running in the user's device, collects GPS and accelerometer data. This data is processed and the most relevant features are extracted by Woorti's Trip Detection module. Features are movement properties that can be used to identify the transport mode. Each mode has a particular set of characteristics that can be found in the sensor readings. For example, the maximum speed of 10 km/h is a property that indicates the walking mode as the most likely. A machine learning algorithm (i.e., random forest [15]) is used to make a prediction based on these features.

To further improve the certainty of the classifier prediction, a P2P network is built. The application running in the user's device establishes contact with the closest users and requests the predicted mode from each one. Consequently, if the classifier is uncertain about the predicted mode, we can use the knowledge from the closest peers to disambiguate between multiple options. Each device is under different conditions (i.e., positioning, susceptibility to noise, handling from the user, etc) and has sensors with different specifications. By using the predictions from multiple devices, we minimize the impact of noisy and incorrect readings on the transport mode decision of each device.

Furthermore, the P2P network allows peers to share knowledge about their trip history, consisting of validated trips. This knowledge includes metrics to evaluate the performance of the classifier (i.e., the relation between the predicted and real mode) and allows the system to compare trips in order to improve the transport mode prediction. Thus, experienced users with a vast trip history can transfer their knowledge to the new users, allowing them to immediately use that information.

1.6 Document Roadmap

This document is organized as follows. In the next chapter, we present an analysis of the related work. Chapter 3 presents our solution, while explaining the architecture and information flow of the system. In Chapter 4 we discuss the implementation details. Chapter 5 shows the evaluation of the solution, with tests made in multiple scenarios. Chapter 6 presents the fundamental conclusion remarks.

2

Related Work

Contents

The detection of travel mode is included in the field of Activity Recognition and provides information that is relevant in multiple contexts such as collecting data to manage traffic and road congestion, the identification of common travel patterns to improve transport infrastructures, the automation of specific settings in smartphone applications (e.g., user's playlists, advertisements) and many more. In this section, we discuss relevant classification solutions and compare them to the objectives and requirements of our work. In Section 2.1 we analyze stand-alone approaches, where the whole detection process occurs in the user's device. In Section 2.2 we analyze the solutions that make use of a remote server to receive the collected data and to run the detection algorithm. The techniques used to minimize the smartphone's battery consumption are discussed in Section 2.3.

2.1 Stand-alone Classification

The improvements associated with mobile devices technology in the last years, specifically hardware components (e.g. CPU, RAM, storage), led to the ability to run more complex and resource-intensive applications in the average smartphone [16]. This is relevant for stand-alone solutions of transport mode detection where the classification algorithm runs in the device itself. The main advantage of this approach is that there is no data exchange between the application running in the user's device and a remote server, allowing the classification to be performed in real-time. The exchange of data in an application that aims at identifying the transport mode in real-time would imply permanent internet connection, which is an unrealistic assumption. Additionally, without needing to transfer data remotely (e.g. user location), the stand-alone solution ensures the privacy of the users.

However, even with the increasing performance of the most recent smartphones, they are still significantly limited when compared with the dedicated computers offered by remote servers (e.g., cloud services). The complexity of the detection algorithm is conditioned by this resource restrictions, impacting the accuracy of the classifier and the number of different modes that it can identify.

The stand-alone solution to identify the transport mode with the highest accuracy (96.8%) that we find in the literature was proposed by Martin et al. [17] using the GPS (sampled every second) and the accelerometer (sampled five times per second). This work explored and evaluated three different methodologies (movelets, k-nearest neighbours and random forests) to distinguish between five travel modes: walking, biking, bus, car and rail. To adapt the classification process to the smartphone limited resources, they also investigated two feature reduction techniques: principal component analysis (PCA) and recursive feature elimination (RFE). They found that the most optimal method was using a random forest algorithm (trained with 10-fold cross-validation) with the 12 most optimal features (10 speed features plus 2 acceleration features). These optimal features were obtained using the RFE algorithm, that recursively removes the weakest features. Weak features are identified with the Gini index, which is

used to find the features that are better at splitting the data in each random forest node. One feature can be individually weak (low accuracy while using it) but important when used within a subset of other features. The RFE takes this into account by evaluating the accuracy of the subsets that don't contain each specific feature and outputting the optimal features that form the most accurate subset [18]. The strongest features, with a significant disparity, are the mean change in acceleration and the 80th percentile speed. They also tested the creation of features with window sizes of 30, 60, 90 and 120 seconds. The 12 most optimal features only contain 90 and 120 seconds features while the first seven correspond to 120 s. This suggests that a higher window size is needed to contain enough relevant transport characteristics. It was claimed that these methods could be utilized in a smartphone without substantial burden on battery life. However, no power consumption data was presented to support this claim. Additionally, their data was collected from only six students with ages between 18-25. This process could be improved by collecting data from a greater number of individuals with a wider set of combined characteristics (e.g., age, gender, professional activity, etc.) to form a more complete dataset with a high variety of users. Furthermore, their evaluation does not include travel segments with less than 120 s, which is a significant factor that influences the high accuracy obtained.

Reddy et al. [19] proposed another stand-alone solution using a decision tree followed by a first-order discrete Hidden Markov Model with data collected from the GPS and accelerometer. The classifier identifies 5 classes: stationary, walking, running, biking and motorized vehicle. The motorized class includes multiple vehicles (e.g. car, bus, train) without making any distinction between them. This approach achieved an accuracy of 93.6% that is justified by this grouping of motorized classes. These 5 classes are simple to distinguish due to the differences in their respective key features (e.g. acceleration and speed). For example, running has a high variance in acceleration (along the 3-axis) when compared to walking or biking due to the impact of the feet hitting the floor. The group of motorized vehicles are easily distinguishable from walking or running by their vibrations and speed. The features of different motorized transports are quite similar and their ungrouping would significantly reduce the accuracy of this solution.

Sauerländer-Biebl et al. [11] investigated and evaluated one stand-alone solution based on fuzzy rules (with the GPS and accelerometer data) presented in [20]. Firstly, they start by performing a single-mode segmentation of the collected data. Stop points (where the user speed is 0 km/h) are used to define the segments. Then, they use a set of rules, that is manually defined through the analysis of test data, to establish a membership degree (from zero to one) to each class. These rules are based on parameters related to speed, acceleration, turning angles, etc. For example, for a maximum speed of 40 km/h they attribute a certainty value of 0.8 to a tram, 0.7 to a bus and 0.65 to a car. Other rules will be applied to further improve the certainty degree towards the correct mode. They obtained a high accuracy (98%) for car trips classification (which was the main focus of their work) but had rather low results

(75%) for the overall classifications. Methods based on fuzzy logic are relevant in the classification of incomplete and uncertain information (e.g., facial pattern recognition, weather forecasting, etc.) which is also the case of transport mode detection [21]. However, to correctly identify modes of transportation with the varied conditions discussed in Section 1.3, it is extremely helpful to use a machine learning approach that is able to adapt and learn as those conditions vary. Das & Winter [22] proposed a more interesting solution that uses a hybrid solution with fuzzy rules and a machine learning algorithm (neural networks) as we will analyze in Section 2.2. Furthermore, the authors don't provide data related to the device's power consumption.

Xia et al. [23] collect the accelerometer and GPS data to identify 5 modes: stationary (stay and wait), walk, bicycling and motorized transport. They used a discrete fast Fourier transform to extract the acceleration frequency-domain features from the accelerometer. Velocity features (maximum, minimum, mean and standard deviation) were obtained from the GPS receiver. To classify the transport mode, they start by applying a filter (velocity threshold) to identify if the user is moving or stationary. Then, they use two classifiers, A and B, based on Support Vector Machines. Classifier A is used when the user is moving and will specify if the transport mode is a bike, motor vehicle or walking. Classifier B is used to distinguish between the different types of stationary, namely: stay, wait(walk), wait(bike), wait(motor). The mode wait represents the case where the user stops for a brief moment in the middle of his travel, indicating possible traffic congestions or traffic lights. To reduce input dimensions for the classifiers, they applied an Ant Colony Optimization (ACO) to filter the weakest features and select the stronger features for transport detection. They obtained an accuracy of 96.3%, however, the experiments were done exclusively with the device in the participants' pockets without analyzing the impact and results for other common device positions (e.g., device in the user's hands, backpack, bag, etc.) and orientations. Regarding the battery lifetime, they considered the power consumption when choosing the sampling frequency and selecting the sensors but did not provide any information related to the application impact on the device's battery. Additionally, this solution does not explore the classification of similar motorized transports (e.g., car, bus, train) and chose to group them all into one class that represents vehicles with a motor.

Chen at al. [24] present Mago, a system that uses the accelerometer and the Hall-effect magnetic sensor to distinguish between 7 classes (stationary, bus, bike, car, train, light rail and scooter). The main contribution of this article is the use of the magnetic sensor to infer the user's transport mode. The earth's magnetic field is distorted by the moving metal components of motorized vehicles (e.g., engine, wheels, gears) that generate alternating magnetic fields while rotating. The effect of the distorted magnetic field has multiple features that can be used to distinguish different vehicle types due to the differences in the mechanical structure. For example, cars have smaller wheels when compared to buses which results in a higher dominant frequency in the power spectrum. Given that the magnetic field decays with distance, in

some cases (e.g., in the center of a big bus) the field strength is weak and there is a need to complement the set of features with the use of the accelerometer. They use the accelerometer to detect vehicle vibrations, particularly the effect of different vehicle suspensions and their effect on the road. Their classification method is divided into two layers: motion detection classifier and transit mode classifier. In the first layer, they use a random forest algorithm to distinguish between a stationary state and a motion state. The second layer uses a neural network model to distinguish between the vehicle types. To mitigate errors in sensor readings they buffer the last 5 results and select the majority of predicted modes. This methodology is highly efficient due to the low power consumption (when compared with GPS based solutions) and high accuracy (94.4%). However, Mago requires a smartphone with a Hall-effect sensor with a sensitivity lower than 0.3 µT/LSB and a minimum sampling rate of 100Hz, which is not currently available in a big portion of devices (e.g., iPhone 6, iPhone 5s, Google Nexus 5, etc.). Additionally, this solution is not able to identify the most common physical activities like walking and running.

2.2 Remote Classification

Given the resource restrictions discussed in Section 1.3, some solutions choose to implement the classification algorithm in a remote server (e.g., in the cloud). In this approach, the application running in the user's smartphone (or dedicated device) is only responsible for data collection. This data is transferred to a remote server where the respective classifier is executed. Obviously, a remote server is able to offer more processing capabilities than a smartphone, allowing researchers to explore and implement algorithms with higher complexity than those running in a stand-alone approach. In remote-based solutions, the main focus is to take advantage of the resources available and increase the number of identified classes without having a significant decrease of the overall accuracy value in order to cover more realistic user scenarios (users commonly use multiple means of transport).

2.2.1 Classification with access to geodata knowledge

In the last years, the geographic information systems (e.g., Google Maps, OpenStreetMap) were able to collect and store geographical data from countries all around the world. This data includes roads, bus routes, bus stops, train rails, real-time location of public transports, etc. The information is commonly available to the public and can serve many uses (e.g., route planning, traffic information, distance measurements, proximity to points of interest, etc.). It is particularly relevant in travel mode detection due to the associations that can be made between the user's location and the corresponding geographical information that can be fetched from the multiple datasets available. Server-based online classification offers the possibility to use this information and define new features related to the proximity to transport

infrastructures that will be used by the classifier.

Stenneth et al. [25] uses additional data that represents bus locations, rail lines and bus stops spatial data (available to the public in real-time) to distinguish between motorized transports (e.g., car, train and bus) while still identifying other modes such as bike, walk and stationary with an overall accuracy of 93.5%. For example, if the user is travelling by bus, the system will verify that the user location corresponds to one bus location and trivially identify this mode of transportation. Given the amount of data that represents the transport network, this approach would be infeasible to reproduce in a stand-alone solution due to the storage, bandwidth and battery restrictions but it's one example of the possibilities that the remote server can offer. However, this solution is limited to the geographical areas that offer transport information in real-time. Additionally, the authors don't provide power consumption information but the exclusive use of GPS data (even with a sampling interval of 15 seconds) might drain the battery lifetime of the device.

Another relevant solution that uses external data (OpenStreetMap) was presented by Biljecki et al. [12] to distinguish between 9 travel modes with an overall accuracy of 91%. This solution is based on a set of manually defined if-then rules, also known as an expert system, and fuzzy membership functions to calculate the association degree between a feature and a class. As we discussed in Section 2.1, the context of transport detection includes a lot of varying conditions which results in many possible combinations that are impossible to manually model. A machine learning approach can use the training data and adapt to new conditions, which is not possible in this approach resulting in classification errors as acknowledged by the authors. Additionally, the use of geodata (e.g., roads, railways, bus lines, metro lines) significantly increases the detection accuracy as it provides relevant information to single-mode segmentation, but it limits the use of the system to certain locations covered by the OpenStreetMap. One interesting contribution from this work is the introduction of a hierarchy of transport modes. This hierarchy consists of 3 layers and each layer is a generalization of the previous one. For example, in the third layer we have the classes ferry and sailing boat, in the second layer they are generalized to boat which is also generalized to water in the first layer. Whenever the algorithm has a low certainty on some class, it returns the generalization in the next layer. The hierarchy of transports allows the system to identify a higher number of modes when there is enough data to distinguish them and still offers accurate results (although less specific) when the algorithm doesn't find different features, which is better than returning an incorrect result.

Das & Winter [22] propose a hybrid knowledge-driven framework based on fuzzy logic and neural networks to adapt to the varied conditions discussed in Section 1.3, which is a problem of solutions based in exclusively fuzzy models. Their solution consists of multiple Adaptive Neuro-Fuzzy Model Inference System (ANFIS) representing each transport class (walk, bus, train and tram). Each ANFIS block contains a set of IF-THEN rules that are used to attribute a binary certainty factor (zero or one)

to the respective class. A certainty factor of zero means that the feature vector doesn't belong to a class. Instead of manually and empirically defining the parameters of the ruleset, this system learns from the training data the best parameter values to define a membership for a given class. After each ANFIS block outputs its result, the highest value (corresponding to one) is chosen. For example, if the user is travelling by bus, the ANFIS block corresponding to the bus class will use its ruleset with the learned parameters to attribute a certainty factor of one to the user's feature vector. Expectedly, the other ANFIS blocks will output a value of zero and the class bus will be chosen. This approach achieves an accuracy of 83% while being able to explain the reasoning process for a given choice and to tolerate noisy data. However, this solution doesn't identify relevant travel modes (e.g. car, bike, running) and its exclusive dependence of the GPS sensor might significantly drain the device's battery. Additionally, it uses features related to the proximity to bus, train and tram network (routes) which limits the system to certain regions.

The knowledge related to the transport network can be successfully used to identify the travel mode as proven by the solutions above. The accuracy of the classification can be significantly improved by using geodata knowledge. For example, if the user is moving through the trajectory of a train railway, the classifier can easily infer his travelling mode with almost 100% certainty. However, regional limitations arise. Some data (such as bus location in real-time) is not covered in most of the cities. Additionally, the overlap of transport networks might result in classification errors. For example, one bus route might intersect with one tram railway. Consequently, this methodology must be complemented by other features obtained from the smartphone sensors. This complementation is particularly needed in the case of solutions that aim at identifying other travel modes without any significant correlation with geographical data (e.g., walk, bicycle). Furthermore, the computation and analysis of the amount of data needed to represent these infrastructures will result in performance issues. As experienced by Stenneth et al., it took them 2 minutes to explore the dataset (with a linear comparison) and create the features (e.g., closest distances to buses, rail trails, bus stops). This performance issues must be addressed in ordered to develop a solution based on geodata knowledge with near real-time requirements.

2.2.2 Classification entirely based in sensor data

The exclusive use of the smartphone sensors to infer the transport mode allows the system to have a better world coverage. These approaches are not dependent on third parties providing enough data for feature selection nor limited to certain cities. The features are extracted only from the built-in sensors, representing the characteristic and universal properties of each mode.

Hemminki et al. [26] used a purely accelerometer-based approach to distinguish between 7 different modes (stationary, walk, bus, train, metro, tram, car) with an accuracy of 84.9%. They developed an algorithm to estimate the gravity component of the accelerometer readings and consequently derive the

component that represents the user motion. Then, they extract a total of 78 features of user movement that are used in a hierarchical classifier. First, the kinematic motion classifier will verify if the user is walking, progressing to the stationary classifier if another activity is detected. The stationary classifier distinguishes between stationary or a motorized transport that is later classified to bus, train, metro, tram or car in the motorized classifier. The kinematic classifier consists of an instance-based classifier combined with a discrete Hidden Markov Model. The stationary and motorized classifiers consist of a voting scheme for classifications obtained with an instance-based classifier. A variation of AdaBoost (adaptive boosting) is used as the instance-based classifier. AdaBoost is a machine learning algorithm that combines the output of weak classifiers, that learn through different subsets of features, into one strong classifier. This algorithm has the disadvantage of being sensitive to noisy data, which in the case of smartphone-based data collection is quite frequent due to the constant variations of the device orientation and interferences related with the handling of the device. Additionally, the authors report a high latency in the classification when the user changes to a motorized transport, which could be solved with the use of other sensors (e.g., GPS).

Zheng et al. [27] used only the GPS data (from GPS dedicated devices) to identify 4 classes (walk, car, bus and bike) with an accuracy of 75%. In this work, they perform the segmentation of the GPS trajectory by assuming that there is a moment where the user stops when changing between modes and that the user walks in the transition of different transports. Thus, a loose upper bound of velocity and acceleration is used to identify the walking segments that are then used to partition the trajectory into the remaining segments. To infer the transport mode, they use a decision tree applied to three features: heading change rate, stop rate and velocity change rate. A graph-based post-processing algorithm is later executed to further improve the detection accuracy, using the knowledge about the common user behaviour. The main idea is to group users' transport change points into nodes and connect them according to the GPS trajectories while assigning to each edge the probability associated with each transport mode.

Prelipcean et al. [28] present Mobility Collector, which is a tracking and annotating framework that collects both GPS and accelerometer data from the user's smartphone and allows him to manually annotate the travel mode. They present a case study where this system is used to automatically detect the user's transportation mode, using the mobility-related data and the respective annotations inserted by the user to train the classifiers. Multiple classifiers (Bayesian Network, Support Vector Machines, Decision Tree, Random Tree and Random Forest) were trained on the collected data using K-fold cross validation (K=10). The obtained results show that the random forest classifier had the highest accuracy (90.8%) while distinguishing 7 modes: car, train, walk, subway, bus, bike and ferry. In this solution, a unique identifier for each user is used as a feature to represent the user's specific behaviour. It is claimed that this feature increases the accuracy of the classification by 6.2%. However, this unique user

identifier will decrease the ability of the machine learning algorithm to identify the transport mode for new users.

Zhou et al. [13] present a GPS based solution that is able to identify 6 modes (walk, subway, bus, car, e-bicycle, bicycle) with a maximum accuracy of 94.4%. The first step is to perform the single-mode segmentation by identifying the transition points through a comparison of the features that occur before and after each tracking point. The region around each transition point (first and last 10 tracking points) is defined as a transition region and it is not used in the travel mode detection. To identify the transport mode in each segment, they use five sets of 24 features that are related with speed, temporal and spatial distribution (relevant for subway detection due to the identification of signal losses), acceleration, stop rates and direction changing rates. These features are used by a random forest algorithm (trained with a user survey) while using the Gini coefficient to evaluate the contribution and importance of each feature to the classification. This approach obtained a rather high accuracy value while distinguishing between six different modes. However, the exclusive use of the GPS sensor has a few problems related to the device's power consumption and the use of the system in weak-signal regions. For example, this system can't distinguish a car travelling on an underground road and a subway.

2.3 Smartphone Power Consumption

The battery life is a relevant aspect in every smartphone application, particularly in the field of travel mode detection due to the energy consumption associated with data collection from the smartphone sensors. In order to assure a complete collection of user's daily travel patterns, the application must be able to run through a normal day of an individual without significantly deplete the device's battery. Otherwise, the user might feel motivated to uninstall the application. Consequently, the choice of which sensors to use is not only based on the ability to detect relevant transport features but also to minimize the power consumption of the application running in the smartphone.

GPS is a position tracking technology that provides the location of a receiver (e.g., incorporated in smartphones) as long as an unobstructed line of sight exists between the receiver and at least 4 satellites. With consecutive tracking points, an application can infer the user's travelling speed. Given that speed features (e.g., maximum, minimum, mean, etc.) form a strong feature vector to distinguish between different travel modes, as proved in [23], many solutions opt to use this sensor (e.g., [12], [13], [17], [25], etc.). Additionally, the GPS also allows the applications to use the user's location to locate relevant geo-locations (e.g., bus stops, bus locations, rail trails, etc.), which is used in [12], [22] and [25] to infer the most probable transport mode. However, GPS is the most power-hungry sensor representing 53% of the smartphone's power consumption in outdoor environments while the accelerometer only consumes 8% in the same conditions, according to the analysis made by Khan et al. [5]. To minimize

the power consumption associated with data collection, some solutions based on the GPS opt to limit its use when the user isn't moving.

Prelipcean et al. [28] use the accelerometer do detect if the user is inside a building and, therefore, not travelling. In this approach, they only activate the GPS when the average of the values collected by the accelerometer within a specific time interval exceeds the defined threshold, indicating that user's movement might be relevant enough to collect GPS data. Thus, the application avoids wasting energy with irrelevant information. Additionally, this work proposes an equidistance tracking that consists of dynamically adjusting the sampling frequency according to the user's speed. This technique defines a distance interval between consecutive points. The trajectory is then defined by a sparser set of tracking points, that is also able to provide the pretended features but without the sampling of unnecessary points.

A different approach to control the GPS data collection was proposed in the work of Reddy et al. [19] that use the GSM sensor to detect when the user is moving, given that it is less power-hungry than the GPS sensor. GSM is a standard protocol for mobile communications. In this protocol, the mobile station (smartphone) communicates with a base transceiver station (BST) that has a given range. When the device is stationary, it is locked into certain cell towers. If the user starts moving significantly, the device will enter into a different area and lock on other cell towers, which triggers the system to restart the collection of GPS data. Thus, this solution minimizes the device's power consumption by only enabling the GPS sensor when there is relevant data to collect.

Other solutions tried to improve battery life with different methods. Martin et al. [17] applied an RFE algorithm to reduce the number of features resulting in a less intensive feature computation. Xia et al. [23] considered the power consumption of the sensors when choosing the sampling frequency given that higher frequencies will result in a faster battery drain. Chen et al. [24] opt to use the Hall-effect magnetic sensor that has a power consumption of 48 mW compared to the 176 mW obtained in the GPS utilization. Some solutions based on a remote server classification, as referred by Stenneth et al. [25], claim to improve the battery lifetime by delegating the detection computation to the respective servers. However, the majority does not address the power consumption associated with the data collection, which represents a big portion of the battery drain. Das & Winter [22] opted to group a few GPS locations into a more coarse-grained time window (i.e., between 1 and 2 minutes) and sent the grouped locations to the server. Thus, the application running in the user device does not need to constantly (e.g., every second) transfer the collected data do the server, which is claimed to improve the battery lifetime.

2.4 Summary

In Table 2.1 we summarize the main characteristics of the solutions discussed in the related work.

The solutions presented that rely on remote classification (see Section 2.2) automatically fail to comply with the real-time requirements due to the delay associated in the establishment of a connection to the main server and the time to transfer the collected data. However, some interesting techniques for feature reduction, classification of travel mode and improved battery consumption were presented. Additionally, relevant information was presented regarding feature selection and the weight of multiple features in the user's movement

Regarding the stand-alone approaches, in the article proposed Martin et al. we have no data to verify the requirements of low battery consumption. However, we assume that this requirement is not fulfilled because they do not implement any sensor managing technique.

Reddy et al. and Xia et al. failed to meet the requirement for the distinction of most common travel solutions (e.g. car, bus, train) because they group the motorized classes. Also, Xia et al. do not provide power consumption data, so we can not verify the fulfilment of low battery consumption requirements.

In the solution of SauerländerBiebl et al., the obtained overall accuracy (75%) is lower than the accuracy required in our solution (90%). Also, we can not verify the fulfilment of low battery consumption requirements because there is no data to evaluate this metric. However, we assume this requirement is not met because there is not any methodology to manage the sensors.

Chen et al. focused on motorized vehicles, so it does not fulfil the requirements for the distinction of some common travel solutions (e.g., walk, run).

Other solutions focused on activity recognition can be found in the literature (e.g., [29], [30], [31]). However, these solutions are more focused on activities such as sitting, standing, upstairs, going downstairs or upstairs, etc. These type of activities ate not in the scope of our work.

Article	Approach	Sensors	Classes	Detection	Geodata	Accuracy
Sauerländer--Biebl et al.	Stand-alone	GPS, accelerometer	5	Fuzzy rules	No	75%
Biljecki et al.	Remote	GPS	10	Fuzzy rules	Yes	91%
Zhou et al.	Remote	GPS	6	Random Forest	No	94.4%
Martin et al.	Stand-alone	GPS, accelerometer	5	Random Forest	No	96.8%
Reddy et al.	Stand-alone	GPS, accelerometer	4	Decision Tree followed by HMM	No	93.6%
Das & Winter	Remote	GPS	4	Fuzzy rules with Neural network	Yes	83%
Xia et al.	Stand-alone	GPS, accelerometer	5	SVM	No	96.3%
Chen et al.	Stand-alone	Hall-effect. accelerometer	6	Random Forest and Neural Networks	No	94.4%
Stenneth et al.	Remote	GPS	6	Random Forest	Yes	93.7%
Hemminki et al.	Remote	accelerometer	7	Expert system and the combination with HMM	No	84.9%
Zheng et al.	Remote	GPS	4	Decision Tree	No	75%
Prelipcean et al.	Remote	GPS, accelerometer	7	Random Forest	No	90.8%

Table 2.1: Summary of the main characteristics from each solution

3

Solution

Contents

In Section 3.1, we describe the background of this work, particularly the decision process behind the initial transport mode prediction, including the detection of a trip, separation of each trip into single-mode segments and the corresponding detected mode of transport.

We start by presenting an overview of our solution in Section 3.2, where we briefly describe the main modules such as the *Trip Detection*, *Trip Validation Manager*, *P2P manager* and the *Transport Mode Determiner*. In Section 3.3, we present the trip validation process and what information is associated with the validations. In Section 3.4, we describe how the devices communicate with each other, the information exchanged and how it is used to improve the prediction. Finally, in Section 3.5 we explain how the output from the other modules is integrated to take the final decision towards the travel mode used by the user.

3.1 Background

DetectP2P imports the Trip Detection module of Woorti, a mobile application used to understand the value of travel time perceived by the user. In this section, we briefly describe how this module works in order to better understand how it can be improved in the following sections.

The Trip Detection module collects the smartphone's sensor data (i.e., GPS and accelerometer) and processes it to extract the most relevant features. These features are evaluated by the random forest classifier to predict the transport mode used by the user.

The GPS is used to obtain the coordinates that represent the user's location. From multiple GPS coordinates, the user's travel speed is inferred by dividing the distance travelled by the time interval. The accelerometer is used to obtain the user's acceleration along three axis.

These raw values, that represent acceleration and speed, are grouped into trip segments with 90 seconds. The raw data from each segment is processed in the Preprocessing module (see Figure 3.1) to calculate the features of the transport mode. These features represent the characteristics of the trip, such as average, maximum and minimum speed, percentage of acceleration values between 0.3 m/s^2 and 0.6 m/s^2, percentage of acceleration values between 0.6 m/s^2 and 1.0 m/s^2, etc. The complete set of 23 parameters that describe each segment can be consulted in the article where Woorti's methods were developed [14].

After processing each segment, the resulting parameters are fed to a random forest classifier that evaluates the segment and stores the resulting list of probabilities correspondent to each possible transport mode. When the user is stationary (i.e., distance and accelerations smaller than a defined threshold) the trip ends, and the post-processing phase begins.

In the post-processing phase, strong segments (i.e., segments with a high walking probability) are used to split the sequence of segments into smaller sequences. These walking segments are easily

identifiable due to the high acceleration modulus (empirically defined). This allows us to identify where each mode of transport sequence starts and finishes. Finally, each sequence is evaluated again and the final mode of transport for each sequence is obtained.

The random forest classifier was previously generated in the training phase, with 537 trips (corresponding to 265 hours) obtained from a group of volunteers collecting data in their daily routine. Those trips were separated into a training and a testing data set to train the classifier. Then, the classifier model was generated and imported to the smartphone application.

3.2 Solution Overview

DetectP2P collects data from the smartphone sensors and evaluates it with the machine learning approach developed in Woorti, where a mathematical model based on training data (i.e., GPS and acceleration metrics) is used to make an initial prediction on the current mode of transport. The validation history of the local user is used to adjust this initial prediction. DetectP2P communicates with the closest devices via Bluetooh to exchange stored user validations and transport mode predictions in real-time. Then, the knowledge obtained from the local and external validations is used to make a prediction based on the route taken. An additional prediction is created by analysing the predictions from the surrounding devices. By combining the three predictions, a final decision is taken.

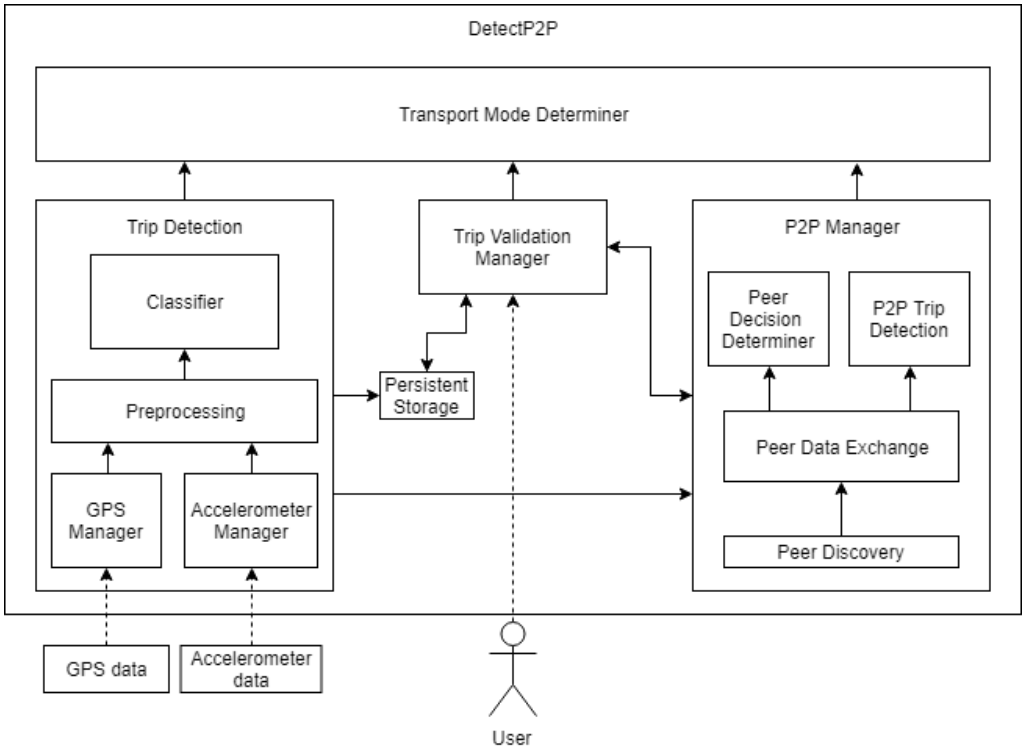

Figure 3.1: DetectP2P architecture and information flow.

A solution that is purely based on a machine learning approach is quite susceptible to prediction errors due to the high dependability on the sensor data collected by the device.

It is important to note that while each device is running the same machine learning model (i.e., random forest classifier), the prediction may vary from device to device due to multiple factors. There are many variables to take into consideration, such as the accelerator sensibility (i.e., device hardware), the timing of the sensor readings, noise, low GPS accuracy, user behaviour (e.g., user swinging his arm introduces additional accelerations), etc. To improve the trip classification, we need to avoid these errors and overcome the classifier limitations in such highly variable conditions.

By sharing real-time information between devices, we can minimize the impact of incorrect readings on the predicted mode. This is done by analyzing the predictions obtained from other smartphones, rather than just relying on the local ones. This is particularly interesting for collective transports (e.g., car, bus, train) that carry multiple passengers, allowing them to use each others' predictions. These modes can be hard to distinguish due to the similar characteristics (i.e., acceleration and speed). While being part of a network of users sharing the same vehicle, this solution allows for extra resistance to the classifier's failed predictions. The application will then perform an analysis of the other devices' classification (i.e., estimate probability for each transport mode), and include their predictions in its own decision.

Additionally, as the users validate trips, their smartphones store them as validations (i.e., sensor data metrics, classifier prediction, actual used mode) that can be used to improve the classifier's decisions. With the possibility of exchanging data between devices, we can pass the knowledge obtained from multiple users' validations. For example, one user that starts using the application can immediately obtain an improved decision if he is travelling with a more experienced user. The experienced user has the history of his validations and the validations from all the users that were connected to him in the past. When the new user receives this information, he can process it and obtain relevant knowledge relative to the performance of the classifier on each mode of transport.

The communication link used to exchange data between devices has a limited range, which means that the application will only share data with the closest devices. This condition ensures that a big portion of the knowledge obtained by communicating with other devices is relative to the surroundings of the user location. As the travel conditions may vary from city to city (e.g., different transports, traffic density, road characteristics, etc.), it is more effective to use data relative to the user area.

When it is required to perform a transport mode decision, DetectP2P collects the prediction from the local random forest classifier, improving it with predictions from other users and the data obtained from all the known validations.

In Figure 3.1 we show the main modules of our application running in each device. The final decision towards the user's transport mode is taken by the Transport Mode Determiner. This module gathers the

output of three other main modules: Trip Detection, Trip Validation Manager and P2P Manager.

Trip Detection (developed in Woorti): Detects the start/end of each trip and makes a prediction based on the smartphone sensors data (i.e., GPS and accelerometer).

Trip Validation Manager Collects user validations, storing information that relates the trip with the classifier's prediction and the real mode validated by the user.

P2P Manager: Responsible to establish connections and share information with the other detectP2P instances within the range of the user. This module takes a decision based on the surrounding smartphone's decisions.

The Transport Mode Determiner module input consists in the results of the three other modules (see Figure 3.1). Both the Trip Detection and the P2P Manager output consist in a prediction where for each possible mode there is an estimated probability of that mode being the correct one. In the case of the Trip Validation Manager, the output is a confusion matrix (i.e., observed mode opposing to predicted mode) that resumes the validations obtained locally, and the complete list of validations from every user.

The confusion matrix represents the probabilistic relation between the predicted and the actual transport mode used by the user, based on the previous trip validations (see Section 3.3). For example, we can obtain the percentage of occurrences where the classifier prediction was *Bus* with the actual mode being *Car*. Therefore, we know which modes are more likely to be confused with the predicted mode and use these values to assist in the final decision (see Section 3.5) when the classifier is uncertain between multiple transport modes.

3.3 Trip Validation Manager

When the Trip Detection Module detects and evaluates a trip, it saves the trip structure (i.e., sequences of segments with the corresponding evaluation) in the device's persistent storage (i.e., local database). Later, the user is able to verify the trip and confirm the correct mode of transport. This validation, representing the real mode of transport, is stored together with the classifier's prediction and the respective trip coordinates in the file system. Coordinates are needed to infer the path of the respective trip, which enables the system to later recognize similar trips with equal conditions.

These trip validations allow us to analyze the performance of the classifier, but the validations from a single user might not be enough to extract accurate metrics. Thus, the validations of the other users are also stored in the local database. Those validations are obtained by the P2P Manager while communicating with other devices, as explained in Section 3.4. In order to avoid repeated validations, each validation is associated with a unique user and trip identifier, generated by the original device where the trip was detected. Thus, devices can share validations whenever they connect to each other while this module filters repeated information.

	Real Transport Mode				
Classifier Prediction	Walking	Bicycle	Car	Bus	Train
Walking	96.1	3.9	0.0	0.0	0.0
Bicycle	3.2	93.0	1.8	1.6	0.4
Car	0.2	2.3	75.2	15.6	6.7
Bus	0.0	0.0	10.5	83.4	6.1
Train	0.0	0.0	3.2	2.1	94.7

Table 3.1: Example of the confusion matrix (in %) obtained from validations.

Additionally, this module creates a confusion matrix (Table 3.1) that summarizes the results of the validations. From the confusion matrix, we can obtain the true positives, false positives, false negatives and true negatives, relative to the classifier's performance. Then, we can infer more metrics to evaluate the classifier, such as the false positive rate, false negative rate and precision. These metrics represent the likelihood of one mode being confused with another one. That relation allows the system to adjust the classifier prediction, as we explain in Section 3.5.

It is important to note that the validations obtained were validated by the local user or by users that travel along the same area because the communication premises require the users to be close to each other (i.e., less than 10 meters) in order to share validations. Thus, the metrics offered by this module are mostly relative to one region. The objective is to focus the knowledge of the system in the area where the user spends more time travelling during his daily routine.

3.4 P2P Manager

In this section we describe the peer-to-peer component of our solution, particularly the discovery of peers (Subsection 3.4.1), communication between devices (Subsection 3.4.2), an alternative method to detect trips (Subsection 3.4.3) and, finally, the classification of a trip based on peer data (Subsection 3.4.4).

3.4.1 Peer Discovery Manager

Before starting to share information between devices, the connections must be established. This is the responsibility of the Peer Discovery Module. It discovers and connects to other devices via Bluetooth [32], a limited range communication link that allows the devices to directly connect with each other as illustrated in Figure 3.2.

At any moment, the user is between three possible states: travelling, stationary (i.e., not travelling), waiting (i.e., travelling but currently stationary). When the user is stationary, DetectP2P keeps trying to discover new peers in order to detect the start of a new trip (see Subsection 3.4.3). In the waiting state, DetectP2P needs to be able to decide if the next state is still (i.e., the trip ends) or travelling (i.e., the trip continues). The predictions of other peers are used to decide which state follows the waiting state

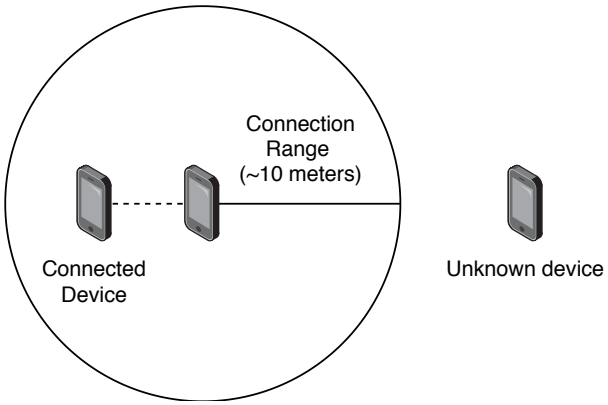

Figure 3.2: Representation of the device connection range.

(see Subsection 3.4.3). In the travelling state, we use the predictions to identify the current mode of transport being used. Thus, we need the discovery process to connect with users sharing the same transport mode and obtain their predictions (see Subsection 3.4.4). Therefore, as the list of peers can be constantly changing (e.g., user's moving in and out of each other's range), the discovery process must be running during the whole life cycle of the application.

Regarding the power consumption requirement, the continuous discovery of peers does not have a significant impact on battery lifetime. Classic Bluetooth consumes approximately 30mA [33], which represents a draining of 1% per hour in a 3000mAh battery.

To keep track of the surrounding users, this module keeps a list of connected peers and the respective connection timestamp. This list is updated whenever one device enters or exists the user's range. When the connection with one peer is established, the devices will be able to share information until one of them disconnects.

3.4.2 Data Exchange

There are two classes of information being shared: real-time predictions and trip validations. Real-time predictions are the local classifier decisions that are broadcasted to every peer when one segment is evaluated. These predictions consist of a list of probabilities associated with each possible transport mode. The trip validations are managed by the Trip Validation Manager and are sent to other peers in order to share the knowledge relative to the classifier's performance. These validations include the classifier prediction, the corrected mode and the trip GPS coordinates. The validation exchange occurs when two users initiate a connection.

Regarding real-time predictions, every time the random forest classifier evaluates a segment (i.e., a trip fragment with 90 seconds), the result of the evaluation is sent to the P2P Manager that broadcasts the data to every connected peer. In Figure 3.3, we can visualize how the predictions are broadcasted.

Each device broadcasts its prediction and receives the predictions of the connected devices.

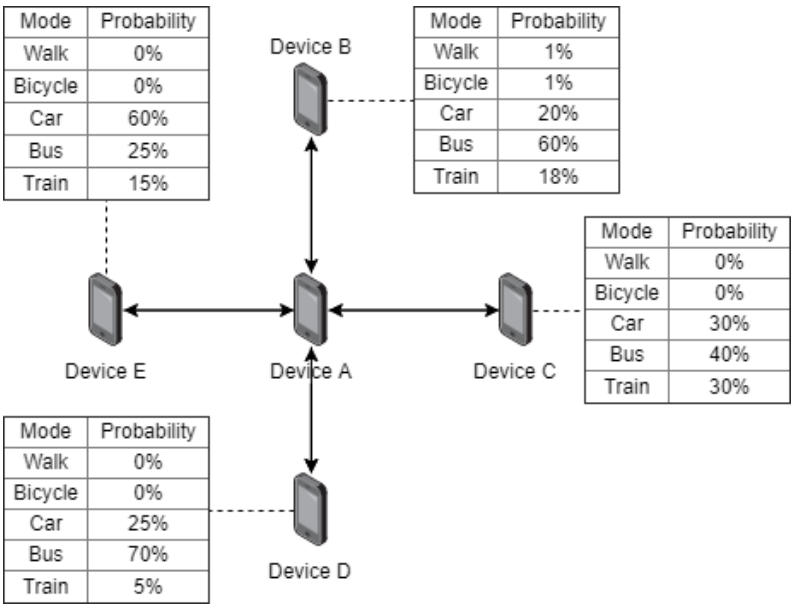

Figure 3.3: Perspective of device A in a network of 5 devices.

In this example, we see the perspective of device A. This device is connected to 4 other devices. Device A broadcasts its classifier prediction to all peers and receives the prediction of each one. When the prediction is received, it is associated with the sender and stored locally, together with all the previous predictions from that sender.

Regarding trip validations, each peer shares all known validations, including the validations performed by the local user and the validations from every device that has ever established a connection. When two devices establish a connection, the validation exchange begins. Each device shares the list of *tripIds* corresponding to every validation stored in its database. Given that each validation is only associated with one unique *tripId*, the other device can filter the validations already acquired and request only the *tripIds* that it does not contain. Then, following that request, all the corresponding validations are sent. The objective of this protocol is to minimize the data being transmitted by avoiding the transfer of repeated validations.

For example, assuming that device 'A' never established a connection with another device, when it connects with device 'B' for the first time, all the validations from device 'B' and the validations that it received from other devices will be sent to device 'A'.

3.4.3 P2P Trip Detection

While the Trip Detection module is the main responsible for detecting the start and end of each trip, the P2P Trip Detection is also able to perform this task when needed. This includes the cases when the Trip

Detection module fails to detect the trip due to lack of GPS signal (e.g., disabled to save battery). This is relevant to collect travelling data when the user wants the device to run on low power consumption (i.e., without activating the GPS).

This module detects the start and end of a trip purely based on the information received from the surrounding peers. The real-time predictions received from other peers, indicating that they are travelling, can be used to detect the beginning of our own trip.

There are four possible scenarios when receiving external predictions:

1. Device A is stationary and connects with a stationary device B.
2. Device A is stationary and briefly connects with a moving device B.
3. Device A is moving and briefly connects with a stationary device B.
4. Device A is moving and connects with device B that is also moving.

By filtering the first three scenarios, we can infer the last scenario, where the device is moving, and recognize the start of a trip. This filter consists in analyzing the predictions from devices with a connection time superior to a threshold of 5 minutes.

The objective of the 5-minute threshold is to distinguish between the devices that only intersect for a small period of time and those that are actually travelling with us. Two vehicles can be close to each other in some situations (e.g., traffic light, stop signs, dense traffic, etc.). However, as soon as the traffic flows, the devices will lose the connection due to the limited range of the communication link. Each prediction is sent when the Trip Detection module evaluates one segment (i.e., every 90 seconds). At least three predictions are required to consider that peer as valid, which translates to 270 seconds. By adding 30 seconds to account for possible delays, we obtain a 300 seconds (5 minutes) threshold.

In scenario 1, there are no moving devices sharing their predictions so there is no indicator of a trip starting. In scenario 2, device B might connect with device A, but the connection time will be inferior to the threshold because the other user will eventually move away and break the connection. In scenario 3, the user is moving but, because there are no moving devices sharing predictions, DetectP2P does not have any information to confirm the start of a trip. In scenario 4, the current device has predictions from a moving device. If the connection time is superior to the threshold, it indicates that the current device is keeping up with a moving device. Thus, DetectP2P can assume that a trip has started.

To detect the end of a trip, we need to infer scenario 1. This is done by finding devices connected for a time superior to the threshold, without broadcasted predictions (i.e., stationary devices). By knowing that we are close to a stationary user for a given period of time, we can assume that we are stationary too. If the count of connected peers decreases to zero while the device is in a travelling state, there is no information to infer that the device is either travelling or stationary. Thus, when there are no peers connected, we also end the trip.

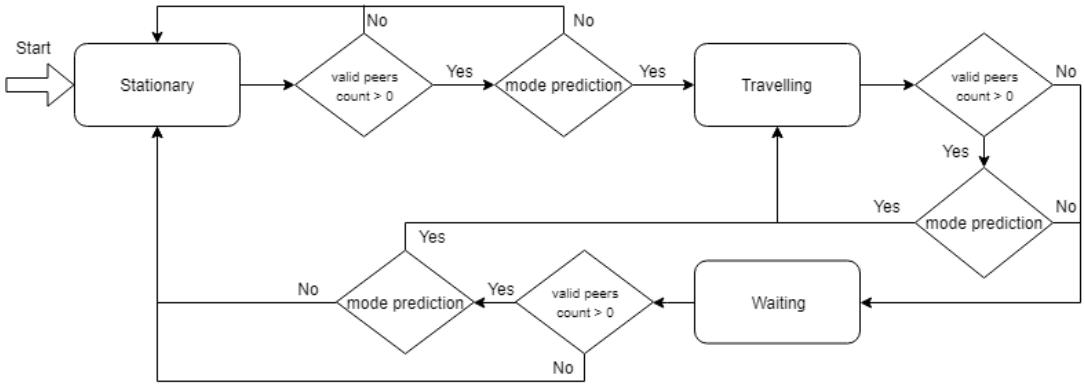

Figure 3.4: P2P Trip Detection state diagram

The scenarios described above result in the state machine presented in Figure 3.4. The state of the user is either stationary, travelling or waiting (i.e., travelling but currently stopped). The transition from state to state is done by testing the count of valid peers (i.e., peers with a connection time superior to the threshold) and their respective predictions. A prediction can indicate that the user is stationary or is in travelling mode (i.e., car, walk, bicycle, bus, train). In the scope of the detection of a trip, we are only interested in testing if the prediction indicates a travelling mode.

3.4.4 Peer Decision Determiner

When the application detects the end of a trip, the decision process in the Peer Decision Determiner begins. The ending of a trip is detected by the main Trip Detection module or by the alternative P2P Trip Detection module.

If the ending of a trip is identified by the Trip Detection module, it splits each trip to a sequence of segments (i.e., trip part), where each trip part corresponds to a travel mode. The sequence limits (i.e., end and begin) are identified by strong segments (i.e., high probability for walk or stationary mode). These strong segments are easily identifiable due to the unique acceleration/speed values.

If the end of the trip is identified by the P2P Trip Detection module (i.e., without recurring to the smartphone sensors), the trip is represented by the same structure (i.e., segment sequences) as it is built from the information passed by the Trip Detection module of another device. Thus, the Peer Decision Determiner takes a decision based on the time intervals that correspond to the limits of the segment sequences. This module analyzes the peer predictions that were received between the beginning and the end of each trip part and makes a prediction based on that.

The first step is to filter the valid peers. To do this, we need the connection timestamp associated with each peer. The connection time is used to filter the devices that are sharing the same transport. If the connection time is inferior to the 5-minute threshold, we will discard the information from that peer.

The communication link properties guarantee that the devices have a small communication range. Given these properties, such a small connection time is not sufficient to assume that the users are sharing the same mode of transport. The smartphone of one user travelling by bus can connect with the smartphone of other user travelling by car, close to the bus. The connection will be established and the devices will exchange predictions. However, the car and the bus will eventually move further away from each other and the connection will end, resulting in a small connection time and, consequently, the predictions being discarded. Thus, the probability of two users sharing the same transport mode increases with the connection time between the two devices. If two users are travelling together, they can compare their predictions to take a final decision. The higher the number of users travelling together, the higher the number of predictions from different devices that can be taken into consideration in the decision.

Each valid peer can have one or more evaluations broadcasted, so the next step is to associate the predictions with each trip part, based on the start and end timestamps. From each prediction sequence, results one single prediction representing the arithmetic mean of the predictions considered for that specific trip part. At this point, we have, for each peer, a list of intermediate predictions. Each intermediate prediction corresponds to the travel mode probabilities associated with a specific trip part.

The next step is to calculate the weighted arithmetic average for all the peers to obtain the decision from the P2P Manager for each trip part. The weight of each peer in the decision corresponds to a confidence factor (CF), calculated with Equation 3.1. The confidence factor is based on the peer's device accuracy, which is calculated by dividing the number of correctly predicted trips (CP) by the total count of detected trips (T). If one of the peers does not have a minimum amount of 10 total trips (i.e., average number of trips collected in a daily routine), that peer is not considered.

$$CF = 1 + (\frac{CP}{T} * 10) \tag{3.1}$$

As previously mentioned, some devices are able to make better predictions due to multiple factors (e.g., GPS accuracy, accelerometer sensibility), so the devices with a higher correct prediction ratio have more weight in the decisions.

Finally, this module returns its output to the Transport Mode Determiner, consisting in one prediction for each trip part from a given trip.

3.5 Transport Mode Determiner

When the application detects the end of a trip, the Transport Mode Determiner is called to take the final decision on the detected mode of transport. This module gathers the output of the other three modules: Trip Detection, Trip Validation Manager and P2P Manager. Each module gives a different contribution:

Trip Detection: Attributes a probability value to each possible transport mode based on the machine learning classifier. This classification is the basis of the local decision and provides the information needed for the P2P Manager to propagate a prediction to other users.

Trip Validation Manager Returns the list of every trip validation and the summary of the local validations in the form of a confusion matrix. This knowledge is integrated into future decisions and increases with users validating trips.

P2P Manager: Assigns a probability value to each possible transport mode, based on the collected peer predictions. The predictions of surrounding users provide extra robustness against the local classifier mispredictions.

The decision process starts with the prediction from the Trip Detection module, corresponding to a probability list with a total of 5 probabilities (one for each mode). Then, the initial prediction is improved by analysing the previous results from the local classifier, which serve as an indicator to how well the device is able to classify each mode in the user's daily conditions. For example, if the local classifier has high false positive rates towards one mode, we use that information to decrease the probability of the respective mode. In the next step, we look into all the validations and verify if the path taken by the current trip already exists in any validated trip. We analyse the transport modes previously taken along that path and merge that information with the starting prediction. If one mode is constantly being used in that specific path, we want to increase the probability of that mode. Finally, we look into the predictions from the peers around us and integrate them into the current prediction to take the final decision.

From the list of probabilities associated with the trip being evaluated, we consider the first, second and third modes with the highest probability. The first, second and third mode with the highest probability are referred to as M1, M2 and M3, respectively. We chose to consider three modes by analysing the confusion matrix obtained in the work that developed this random forest classifier [14]. Those results show that each mode is often confused with two other modes. For example, 5% of the train classifications are actually car trips and another 5% correspond to bus trips, but the train is never chosen when walking or riding the bicycle.

Algorithm 1: Adjustment of the classifier decision

Input: T, confusion matrix of the local validations, T[classified][observed]
D, map with probabilities by mode
C_1 , count of occurrences for M_1
C_2, count of occurrences for M_2
C_3, count of occurrences for M_3

1 $P = T[M_1][M_2] + T[M_1][M_3]$
2 $F_i = C_i/(C_1 + C_2 + C_3)$
3 $D[M_i] = D[M_i] - P/3 + F_i * P$

In the first step, we take into consideration the false positive and the false negative rates obtained

from the local validations. These values reflect the performance of the classifier while performing predictions with the data obtained by the local hardware, on conditions specific to the area of the user. In Algorithm 1 we describe the adjustment of the classifier prediction according to the validation history of the user. The input needed to perform the adjustment corresponds to the local confusion matrix (T) calculated by the Trip Validation Manager, the probabilities map (D) for each mode generated by the Trip Detection module and the count of occurrences for each mode. From the confusion matrix, we obtain the percentage of occurrences (P) where M2 and M3 were incorrectly identified as M1. Then, we decrease the probability of the three modes in equal portions (i.e., a third of P) and distribute P according to each mode frequency.

At this point, the classifier prediction is adjusted by Algorithm 1 with the data obtained from the trip validations. The next step is to iterate through all the trip validations (i.e., local and external validations). We compare the trip coordinates with the coordinates of each validated trip. We consider that there is a match between two trips when at least 80% of the coordinates from the trip being evaluated are found within the trajectory of the validated trip. A 20% error margin is given to account for accuracy errors and differences in the trip start/end point (e.g., one trip might have more coordinates because it was detected earlier). To compare trips, for every coordinate (C) in the trip being evaluated, we check if it is between any two points (A and B) that belong to the validated trip, while taking into consideration a margin for the GPS accuracy (ACC) of 13 meters, which is the maximum deviation found in the analysis of GPS locations with Android [34]). To verify if point C is between A and B, the following equation is tested.

$$distance(A, C) + distance(C, B) < distance(A, B) + ACC \tag{3.2}$$

When there is a match between the validation and the current trip, we update the count of occurrences associated with each considered mode of transport. The objective is to know the number of occurrences by mode, through the path representing the current trip. This metric is relevant because it suggests which modes are more likely to be used along with those coordinates. For example, train validations contain coordinates specific to the train railways. If the trip contains coordinates that match the train path, it strongly suggests that the travel mode is the train. Additionally, there are also specific routes for buses and bicycles, representing relevant knowledge that can be used to further improve the decision.

After calculating the number of events for each mode of transport along that path, we attribute a score that consists in the number of occurrences of the respective mode divided by the total number of occurrences. Thus, the most used mode in that path will have the highest score. The last step is to merge the adjusted initial prediction, the path score and the prediction from the P2P Manager (i.e., predictions from the surrounding peers), by calculating the arithmetic average of the three scores.

Finally, we have the final prediction consisting, again, in a list of probabilities associated with each

mode. To take the final decision, we look at the count of peers that were connected to the device during the respective trip. The number of peers that we are able to connect is directly related to the transport mode, as public transports (e.g., bus, train) can carry more passengers. Therefore, if the count of peers is superior to 4, we choose the public transport with the highest probability. In the case where we have transport modes with a small probability difference (i.e., smaller than 5%), we choose the mode that was most used by the user according to the local validations.

3.6 Summary

In this section, we present the DetectP2P architecture and describe the role of each component in the classification of users' travel mode.

We see that the Trip Detection module generates an initial prediction based on the GPS and accelerometer data, by using the random forest classifier. This part of the solution is based on the Woorti application, which is used to classify users' trips in a stand-alone approach.

Then, we explain the Trip Validation Manager. Particularly, how the user is able to validate previous trips, what information is stored in the persistent storage and what knowledge can be obtained from this validations.

As the core of our solution is based on the exchange of information between peers, we proceed to explain the whole P2P Manager. This module includes four submodules: Peer Discovery, Peer Data Exchange, P2P Trip Detection and the Peer Decision Determiner. We start by explaining how the Peer Discovery discovers the surrounding peers and how it manages the connections to other devices. Then, we describe which information (i.e., validations and predictions) is shared with other users and how it is shared with the network by the Peer Data Exchange. As an alternative to the high power consuming techniques used in the main Trip Detection module, we present the P2P Trip Detection where the detection of trips can be made by analysing the information obtained from the surrounding devices. Then, we describe the Peer Decision Determiner, where the predictions received by the surrounding users in specific time intervals are used to calculate a new prediction.

Lastly, we describe the Transport Mode Determiner. This module receives the initial prediction from the Trip Detection. Then, it makes a request to the Trip Validation Manager and receives the confusion matrix of the local validations and the list of all the validations. The confusion matrix is used to adjust the local classifier prediction, while the list of all validations (i.e., from all users) is used to attribute a score to each mode based on the validations made on the same geographical trajectory. The final step is to integrate all the predictions into the chosen transport mode.

4

Implementation

Contents

In this chapter, we discuss the implementation details of the DetectP2P application. In Section 4.1, we describe the extensions made to Woorti's Trip Detection in order to integrate it in DetectP2P. In Section 4.2, we provide the implementation details of the trip validation mechanism, particularly how trips are saved to the persistent storage. The communication between devices is explained in Section 4.3, where we address the connection protocols and the exchange of data. In Section 4.4, we explain the mechanism that we use to detect trips without using smartphone sensors. The calculation of the prediction based on the data from surrounding peers is explained in Section 4.5. Finally, we describe the process of taking the final decision, that aggregates all the information, in Section 4.6.

4.1 Trip Detection Module

The *Trip Detection module* was adapted from the solution developed in Woorti for the Android operating system, with two extensions to include its output in the information flow of our application. As explained in Section 3.1, this module makes a prediction based on the sensor data while using a random forest algorithm. While in the original solution this prediction was considered the final decision, in this work we only use it as the starting point.

DetectP2P requires a permanent exchange of mobility predictions with the surrounding users. When a segment of the trip (i.e., 90 seconds of sensor data) is completed, that segment is evaluated. Then, this module invokes the P2P Manager in order to broadcast the respective evaluation.

Additionally, in the original solution, whenever the module detects the end of the trip, it saves the trip data in the persistent storage with the respective mode of transport. In our solution, instead of storing the detected mode of transport, we store the whole dictionary of probabilities generated by the random forest classifier. This dictionary consists of one probability value associated with each mode of transport.

4.2 User Validations

When a trip ends, it is stored in the file system of the device. Then, it is available to be validated by the user. The validation process consists in associating the real mode of transport used with the corresponding trip. In Figure 4.1 we present a diagram that represents the entities used to handle trip validations. There is one unique *ValidationManager* that contains multiple instances of *ValidatedTrip*, corresponding to the validations collected. Each *ValidatedTrip* contains multiple *TripCoordinates* that represent the GPS coordinates collected during the trip. A *ValidatedTrip* entity contains the following attributes:

userId: Unique and permanent *String* generated by the operating system to identify each user.

tripId: Unique *String* generated when the trip is validated by the *generateTripId* method.

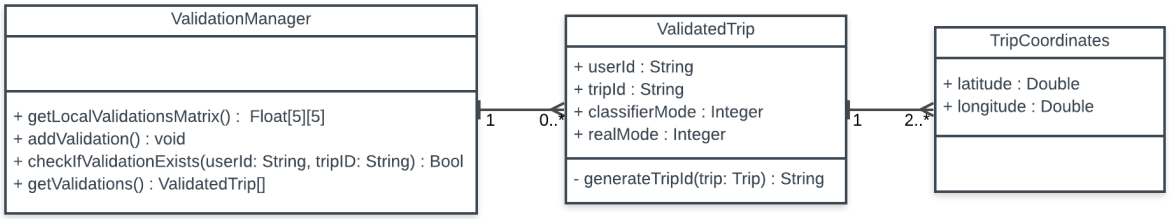

Figure 4.1: UML diagram representing the trip validation entities

classifierMode: Integer that identifies the mode predicted by the classifier.

realMode: Integer that identifies the real mode validated by the user.

tripCoordinates: List of all the GPS coordinates associated with the trip, representing the geographical path taken.

When the trip is validated by the user, the *tripId* is created to avoid the collection of repeated validations. To generate a unique representation of the trip, we instantiate a new *string* and append the following trip data: start timestamp (ms), end timestamp (ms), an average of the absolute acceleration values and distance travelled (m).

The *ValidationManager* is a singleton responsible to handle the storage and access to the validations. The most relevant aspect in this entity is the storage of the validations. As peers are constantly sharing all the validations known to them, we need to assure that there are no repeated trips in the local storage. Therefore, we use the *userId* and *tripId*, generated by the device where the trip was validated, to identify each trip.

Given that the operations that we require consist in fetching the *ValidatedTrips* for specific *userIds*, searching for *tripIds* and storing new validations, the SQLite database is used to store the validations in the persistent storage. The SQLite library is quite simple to implement (i.e., no need for configurations) and allows efficient queries to the data that we need to fetch (i.e., ValidatedTrip table) by using two primary keys: *userId* and *tripId*.

When the *P2P Manager* receives one or more validations from a peer, it forwards the data to the *ValidationManager*. For each validation received, the method *addValidation()* is called and verifies if the trip is already in the database by calling *checkIfTripExists()*. If the trip is repeated, the new validation is discarded. Else, the trip is added to the persistent storage.

4.3 Communication between devices

As explained in Section 3.4, our solution requires close-range communication between devices. The technologies that offer wireless communication between Android devices over short distances are Wifi-

Direct and Bluetooth.

Wifi-Direct has a theoretical range of 200 meters [35], while Class 2 Bluetooth (i.e., most found in mobile devices [36]) has an expected range of 10 meters [37]. If the application uses Wifi-Direct, it will establish connections with users travelling in other vehicles. In this work, we prefer a lower communication range because the objective is to communicate with devices sharing the same transport mode.

Regarding power consumption, Bluetooth is the better option due to the low-energy signal used to exchange data [35], which is an important factor considering the requirements of our application related to the battery lifetime. Additionally, Wifi-Direct requires the devices to establish P2P groups, where one device is the *Group Owner* and the other devices act as clients. When the *Group Owner* disconnects, the group is disbanded and the other clients have to negotiate a new *Group Owner*. In the scope of this work, users are frequently moving in and out of each others' range. For this reason, Wifi-Direct would require a more complex protocol to handle group negotiation in an environment where users are constantly moving and can belong to multiple groups.

Due to the reasons above, we chose to implement the communication with the Bluetooth technology. To do this, we used the *Android Multi Bluetooth Library* [38], an Android library that simplifies the discovery, connection and data exchange between devices.

Before starting to communicate with other instances, each device must select its operating mode, either server or client mode. Server devices advertise themselves and wait for clients to discover them and initiate the connection. In this work, all devices can play the server and client role, otherwise, two instances running in the same mode could not connect to each other. Thus, each device runs in server mode while still running the discovery process.

Whenever two instances discover each other, they need to negotiate who waits and who starts the connection. The negotiation process is decided by generating a random integer and including it in the device name (i.e., the id advertised by each device). Thus, each device advertises himself with the name: *detectP2P* followed by a random number (e.g., detectP2P_1000). The instance with a higher value corresponds to the server and will wait for the other instance to initiate the connection by calling the method *createClient(mServerAddressMac)* provided by the library, where *mServerAddressMac* corresponds to the MAC address of the server. This method will open the socket connection between both devices, allowing for any message to be exchanged.

We store the connection information in the P2P Manager in the form of a Map structure, where for each peer name (e.g., detectP2P_0506) we store the respective MAC address that is needed to transfer data.

As explained in Subsection 3.4.2, our solution requires the exchange of two types of messages: validations and predictions. In the following sections, we explain the implementation details for the message exchange.

4.3.1 Validation Exchange

The exchange of validations occurs when the connection is established, with a simple protocol consisting of synchronous messages. The client device initiates the exchange by sending a list of *tripIds* that correspond to every validation that he has stored. Remember that each validated trip has a unique *tripId* generated by the device where the trip was detected and validated. Thus, to avoid sharing repeated validations, the other device will iterate those *tripIds* and request the *tripIds* that he does not have. Then, a list containing the serialized *ValidatedTrips* (see attributes in Section 4.2) associated with the requested *tripIds* is returned. Following this exchange, the inverse process begins, where the server device sends its known *tripIds* and the client filters and receives the new validations. At this point, both devices contain exactly the same validations.

4.3.2 Prediction Exchange

Every time one device evaluates one segment, the Peer Data Exchange module broadcasts the prediction to every connected peer. Therefore, the exchange of predictions is an asynchronous process.

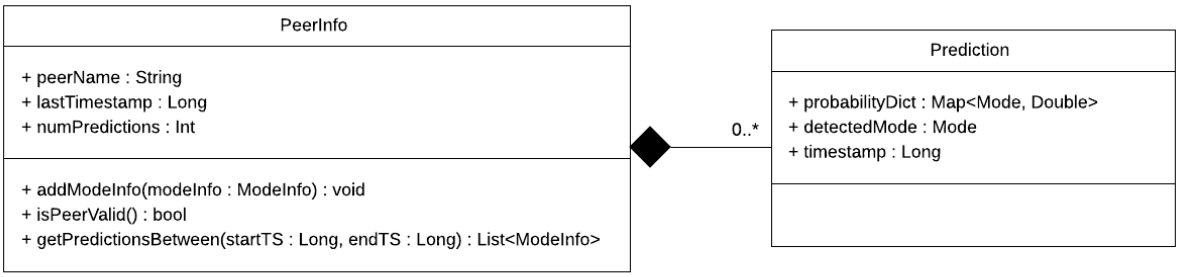

Figure 4.2: UML diagram representing the peer and its predictions

The P2P Manager is responsible to manage and store the peer predictions. We use a Map structure to associate the peer name with the *PeerInfo* class. In Figure 4.2, we present the relation between *PeerInfo* and *Prediction*. *PeerInfo* represents each peer, containing its name, the timestamp of its last prediction, the number of predictions and a composition of the class Prediction. Each Prediction represents the evaluation performed by the classifier of the other device, containing the probability dictionary (i.e., a probability associated with each mode), the detected mode (i.e., mode with the highest probability) and the timestamp calculated when the prediction was received.

The Peer Data Exchange module (see Section 3.2) is constantly waiting for messages from the other devices. When this module receives a prediction from other device, we use its instance name to fetch the *PeerInfo* class from the Map structure or create a new entry if it does not exist yet. Then, the method *addModeInfo* is called to store the association between the prediction and the respective peer. This

method updates the peer's *lastTimestamp* with the prediction timestamp, increments *numPredictions* and associates it with the prediction to keep track of their order.

4.4 P2P Trip Detection

This module (shown in Section 3.2) uses the predictions received to manage the current state of the user. Recall that (as mentioned in Subsection 3.4.3) there are three possible travel states: stationary, travelling and waiting. The state is updated when there is enough information to infer the new state of the user.

When the application is initiated, one instance of the P2P Trip Detection is created by the P2P Manager. This instance defines one Handler that is responsible to manage the queue of a background thread. Then, we create the tripDetectionP2P Runnable, an interface that allows the trip detection code to run in its specific thread. We use the Handler to dispatch the Runnable into a background thread every 90 seconds, which is the minimum amount of time between predictions. Thus, if at one point in time we do not have any prediction to infer the next state, we need to wait for the other peers to send their new predictions which takes approximately 90 seconds (as explained in Subsection 3.4.2).

Every 90 seconds, a *tripDetectionP2P* routine begins its execution. Here, we fetch the last predictions by iterating through the valid PeerInfo instances (i.e., peers connected for more than 5 minutes) and aggregating the results of the *getLastPrediction* method. From the resulting list of predictions, we remove those that are older than 90 seconds as we are only interested in the predictions received between the current timestamp and the last execution of this routine.

Then, after obtaining the last predictions, we perform the following actions depending on the current state:

- **STATIONARY** If there is any prediction with a detected mode of transport, the state is updated to *TRAVELLING*.

- **TRAVELLING** If there is any prediction that detected the *still* mode or if there are no predictions, the state is updated to *WAITING* .

- **WAITING** If there is any prediction that detected a mode of transport, the state is updated to *TRAVELLING*. If there is any prediction that detected the *still* mode or if the count of predictions is zero, we update the state to *STATIONARY*.

4.5 Peer Decision Determiner

When DetectP2P detects the end of a trip, this module receives a request to output a decision between two timestamps, corresponding to the start (*startTS*) and end (*endTS*) of a trip part.

It starts by iterating through the PeerInfo instances and filtering the valid peers with a connection time superior to the 5-minute threshold. For every valid peer, the Peer Decision Determiner fetches the predictions received between the *startTS* and *endTS* by calling the *getPredictionsBetween* method (see Figure 4.2). The average of those predictions is calculated to obtain the peer prediction for the period considered.

Then, we use Equation 3.1 (see Subsection 3.4.4) to assign a weight to each prediction and calculate the weighted average of all the peers. The variables needed to obtain the weight of a peer (i.e., correct predictions and total predictions) are obtained by making a query to the database for the peer Id, obtaining its validations. and calculating the respective values. Finally, we have one final prediction from the peers and return it to the Transport Mode Determiner where the final decision is taken.

4.6 Transport Mode Determiner

When the main Trip Detection module (see Section 3.1) or the P2P Trip Detection module (see Subsection 3.4.3) detect the end of a trip, the trip structure is passed to this module where the process to evaluate the mode of transport begins. The trip structure consists of a list of trip parts where each part corresponds to a mode of transport. Each trip part is represented by the respective start and end timestamps. If the trip end was detected by the Trip Detection module, the random forest evaluation for each part is included in the trip structure.

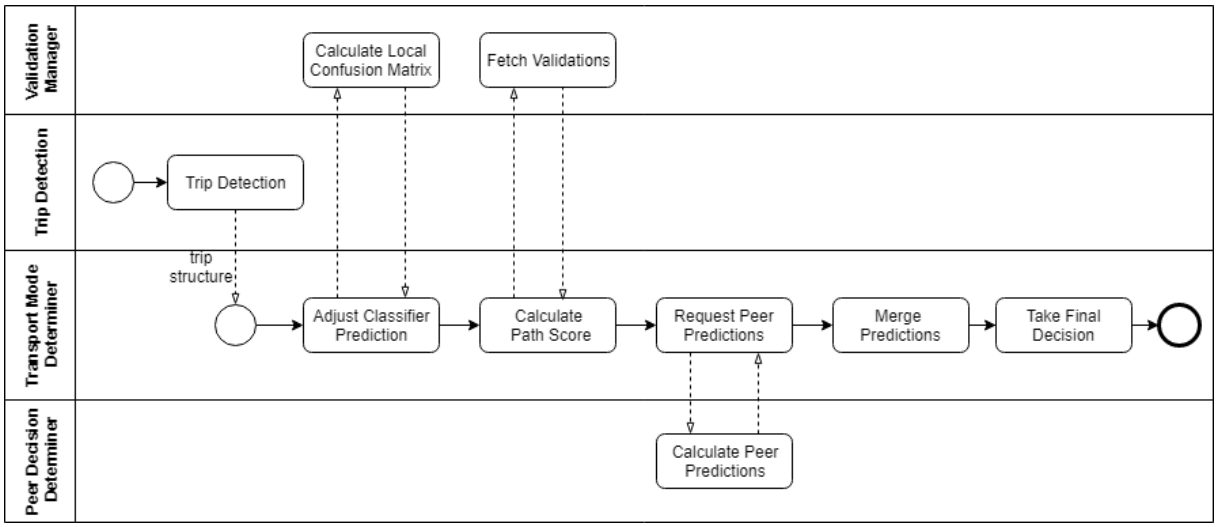

Figure 4.3: DetectP2P decision process

In Figure 4.3 we see the whole decision process and the tasks that are performed to obtain the final transport mode. After receiving the trip structure, the random forest evaluation (when available) is adjusted with the confusion matrix calculated in the validation manager (see the adjustment details in Subsection 3.4.4). The following task is the calculation of the path score for each mode by comparing the trip path with similar paths from the validations.

To calculate the path score, we need to compare the trip being evaluated with every validation stored in the database. Given the high count of validations that each user might have stored and the requirements for real-time performance, this task must be efficient and minimize the time spent iterating through validated trips with a different route. Thus, when fetching validations, we look at the start and end locations. As we require at least 80% equivalent coordinates (see Subsection 3.4.4) to assume the similarity between two trips, we compare the start and end location of the current trip with the start location of the validated trip. If either the start or end location validate the following formula, we fetch the validation from disk to perform a complete comparison.

$$tripsDistance < pathLength * 0.8 \tag{4.1}$$

In Equation 4.1, *tripsDistance* represents the distance between the start/end point of the current trip and the validated trip. The value of *pathLength* represents the length of the current trip. If the *tripsDistance* for both the start and end location is larger than the length of the path, it is impossible for the trips to be similar and we do not need to cache and iterate that validation. The resulting validations are then compared in-depth with the current trip to calculate the path score.

With the adjusted prediction and the path score, we request the prediction from the Peer Decision Determiner and merge the three results into a single prediction. Finally, the task to take the final decision is performed by analysing the peer counts and the modes most used by the user.

4.7 Summary

This chapter describes the implementation details that are needed to correctly develop DetectP2P into an Android device. We start by explaining the extensions made to Woorti's trip detection. Then, we describe how the data is stored when the user validates a trip and the techniques used to guarantee that each device stores one unique representation for each validation. Regarding the peer-to-peer communication, we explained how the devices are discovered and how the connection is initiated. A simple protocol was implemented to efficiently exchange trip validations when the connection is established. Then, we show how the predictions are shared and stored while the communication channel is opened. We describe how the trip detection based on peer predictions is implemented and how the Peer Decision Determiner takes a decision with the predictions of the surrounding devices. Finally, we describe

the tasks executed by the Transport Mode Determiner that start with the ending of a trip and finish with a transport mode decision.

5

Evaluation

Contents

5.1 Evaluation Methodology

To evaluate DetectP2P's accuracy towards each mode of transport, we need to test it in a real scenario where users travel together, exposed to the same travelling characteristics, and analyse the impact that the interaction between their devices has in the detected mode of transport. Although, DetectP2P can be used in multiple scenarios and the results obtained vary with numerous factors such as travelling alone, travelling with other users, the number of validated trips, the route taken, etc. Thus, to analyse how the application performs on these scenarios we perform the following evaluations:

- Accuracy of the local decision (i.e., random forest classifier) and its variation with users' validations.
- Tendency of the users to repeat trips.
- Processing time to calculate the path score.
- Comparison between local and peer predictions.
- Accuracy of DetectP2P with a group of users travelling together.

We requested 5 people to use DetectP2P for two weeks to collect enough data from their routines. This evaluation was made with five Android devices: Xiaomi Mi A2, Xiaomi Mi A1, Samsung J3, Motorola XT1068 and Huawei Y6. These devices have different hardware from different manufacturers.

Our evaluation was divided into two phases. In phase 1, the users were asked to use DetectP2P in their daily routines, while validating trips at the end of the day. In this phase, the users are travelling alone and only the decision from the local classifier is considered. The objective was to collect their validated trips to obtain information about the performance of the classifier on their routines and knowledge relative to the geographical paths taken. Then, in the second week, the users were asked to travel together allowing us to collect the data that represents how the system behaves and evolves in a real scenario. In this phase, the devices share the validations previously obtained and send predictions in real-time, allowing each DetectP2P instance to include more information in its decision.

5.2 Adjustment Process

We start by evaluating how the application performs when the user is travelling alone. In this situation, the final decision is based in the local classifier (i.e., Woorti's decision) and it is adjusted with the local validations (see Algorithm 1 in Section 3.5). The local validations relate the behaviour of the classifier with the preferences of the user (i.e., preferred modes, conditions of the routes taken, etc.). Thus, as the user validates trips, the classifier prediction is adjusted according to his validation history.

To evaluate this scenario, we considered the trips collected by one of the users (userA) in the first week, corresponding to a total of 105 trips, particularly: walk (64), car (24), bus (5), train (7), bicycle (5). In Table 5.1, we see the confusion matrix that represents the results from the random forest classifier while evaluating the 105 trips. With the collected trips, we proceeded to simulate the adjustment process (see Algorithm 1) for different sets of validations as seen in Figure 5.1 to compare the results obtained with multiple sets. The initial accuracy values correspond to the results obtained when there are no validations while the final values correspond to the results obtained when all the validations were considered. We considered the validations from userA by their chronological order, obtaining the following sets:

	Classifier Mode				
Real Mode	Walking	Car	Bus	Train	Bicycle
Walking	61 (95.3%)	0 (0%)	0 (0%)	0 (0%)	3 (4.7%)
Car	0 (0%)	14 (58.3%)	10 (41.7%)	0 (0'%)	0 (0%)
Bus	0 (0%)	1 (20%)	4 (80%)	0 (0%)	0 (0%)
Train	0 (0%)	1 (14%)	1 (14%)	5 (72%)	0 (0%)
Bicycle	1 (20%)	0 (0%)	0 (0%)	0 (0%)	4 (80%)

Table 5.1: Confusion matrix obtained from userA

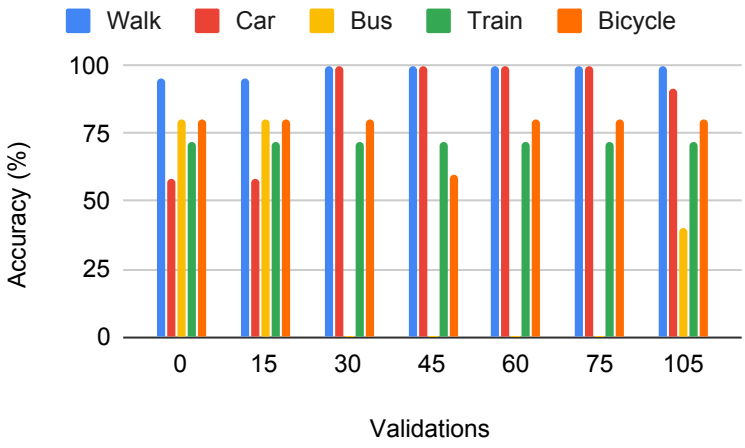

Figure 5.1: Evolution of the accuracy with the local validations.

1. **15 validations** - 11 walk (classified: 11 walk), 4 car (classified: 4 car);

2. **30 validations** - 21 walk (classified: 20 walk, 1 bicycle), 9 car (classified: 4 car, 5 bus);

3. **45 validations** - 31 walk (classified: 29 walk, 2 bicycle), 12 car (classified: 7 car, 5 bus), 2 bus (classified: 2 bus);

4. **60 validations** - 40 walk (classified: 38 walk, 2 bicycle), 18 car (classified: 12 car, 6 bus), 2 bus (classified: 2 bus);

5. **75 validations** - 50 walk (classified: 48 walk, 2 bicycle), 23 car (classified: 15 car, 8 bus), 2 bus (classified: 2 bus);

6. **105 validations** - 64 walk (classified: 61 walk, 3 bicycle), 24 car (classified: 15 car, 9 bus), 5 bus (classified: 4 bus, 1 car), 7 train (classified: 5 train, 1 car, 1 bus), 5 bicycle (classified: 4 bicycle, 1 walk);

Figure 5.1 shows the accuracy (in %) obtained while using the *Trip Detection* decision and adjusting it according to different sets of validations. In the first set of validations, the accuracy values remain unchanged because there are not enough validations to perform the adjustment. With the validations considered, the false positive rate is 0 for each mode because every classification was correct. Thus, Algorithm 1 distributes a probability of 0 to every mode which does not change any prediction.

In the second set, we see an increase in the walk and car modes while the bus significantly decreases. This is explained by the preferences of the users towards the walk and car modes. In the three occurrences where the classifier chose the bicycle instead of the walking mode, the system looked into the bicycle false positives (4.8%) and distributed that probability according to the walking and bicycle frequencies. The same event occurs in the car trips that were incorrectly identified as bus, with the bus classifications being adjusted to car due to the observed user preferences. However, the bus trips are incorrectly adjusted because the system highly favours the car mode at this point.

Between the third, fourth and fifth sets of validations there are no variations in the accuracies obtained, except for a 20% decrease in the bicycle mode. For 60 validations we obtain a false positive rate of 6.5% towards the bicycle and that value is distributed according to the frequencies of each mode, which is enough to incorrectly adjust one of the bicycle classifications. However, in the fourth and fifth set, the false positive rate towards the bicycle drops to 4% and 5%. These values are not enough to change any of the bicycle predictions, therefore, the bicycle accuracy returns to its initial value.

When the total of 105 trips is considered, the accuracy for car drops to 91.6% due to the bus and train trips that are added to the validations. In this situation, the frequency of the car drops to 66.7% which is enough to maintain 2 incorrect bus predictions in car trips. Consequently, we also observe an increase in bus accuracy to 40%.

Here, we can see how the predictions are adjusted according to the preferences of the user. The most used modes are favoured by the adjustment process while taking into account the false-positive ratio of the mode predicted by the classifier. This process significantly affects the modes used with a smaller frequency. However, in a P2P environment, the system can compensate this factor with additional techniques as we prove in the following sections.

5.3 Path Score

In this section, we consider the total amount of 316 trips collected by 5 users. The objective of this evaluation is to understand how the increasing amount of validations provides relevant data to infer the mode of transport and analyse the impact, in terms of processing power, that the calculation of the path score has on the trip detection.

5.3.1 Analysis of the Path Relation

In this evaluation, we compare the routes of all collected validations to observe the number of trips that would be improved by integrating the path score in DetectP2P's final decision. Every repeated trip (i.e., a trip with an equivalent route found in the validations) has an increased probability towards the mode validated in the original trip. Within the repeated trips, we also analyse trips repeated from peers where the original trip was validated by other user.

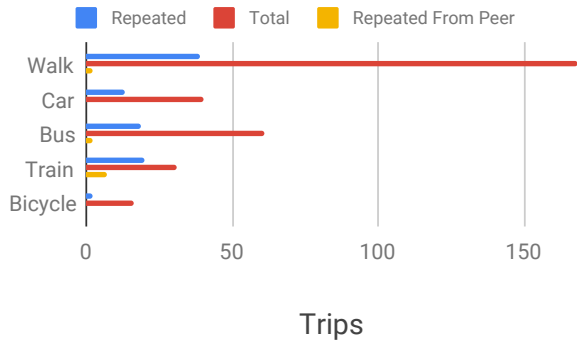

Figure 5.2: Visualization of the amount of trips with equivalent routes.

In figure Figure 5.2 we show the portion of trips with equivalent routes in comparison with the total amount of collected trips. By iterating through the total of 316 trips, we searched for trips with equivalent paths. For each trip, we only compare it with the previously iterated trips. For example, to verify if the third trip is repeated, we compare it with the first and second validated trip. We obtained a count of 93 (29.4%) repeated trips, where 39 correspond to walk, 13 to car, 19 to bus, 20 to train and 2 to bicycle. Among the repeated trips, there are 11 (2.8%) occurrences where an equivalent trip was obtained from a peer (i.e., external validations), while the remaining events correspond to equivalent trips found within the local validations.

5.3.2 Processing cost

Given the time requirements for the output of a transport mode decision, we need to analyse the computation cost to obtain the path score. This evaluation was performed on a Xiaomi Mi A2, featuring

a Qualcomm Snapdragon 660 and 3GB of RAM. To perform this test, we considered multiple sets of validations (50, 100, 300, 600, 1000, and 2000) and a trip with 360 locations (i.e., approximately 30 minutes). Each set contains small trips (50%) with 120 locations (10 minutes), medium trips (40%) with 300 locations (25 minutes) and large trips (10%) with 720 locations (60 minutes).

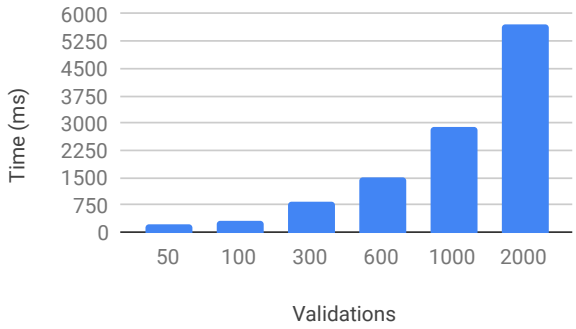

Figure 5.3: Execution time (ms) to calculate the path score.

Figure 5.3 shows how the execution time needed to attribute a path score increases with a higher amount of validations. We do not see a linear growth of the execution time because each validated trip has different coordinates. Each validation is discarded when the trip comparison reaches the threshold of distinct coordinates (20%). Thus, the time needed to discard each validation is different across the set. Even considering the highest value tested (i.e., 2000 validations), we obtained a processing time inferior to 6 seconds which is acceptable considering our requirements for the output of a decision in 30 seconds.

The number of validations considered in this evaluation does not necessarily represent the total amount of validations in the device. As explained in Section 4.2, we optimize the fetch operations and retrieve only the relevant validations (i.e., validations close to the location considered).

5.4 P2P Evaluation

In this evaluation we test the transport mode detection in a P2P environment where the users travel within a group and their devices communicate with each other during the trip. The objective is to see how the classifier predictions vary between different devices and evaluate our solution, with the contribution from every module, in a scenario where a group of users are travelling together.

5.4.1 P2P Manager Decision

Figure 5.4 compares the local prediction (relative to the real mode used) of one device with the prediction obtained from the P2P Manager, where the weight of each peer is based on the trips collected in phase

1 (i.e., users travelling alone). Each value in the x-axis represents a single trip without any relation with the others.

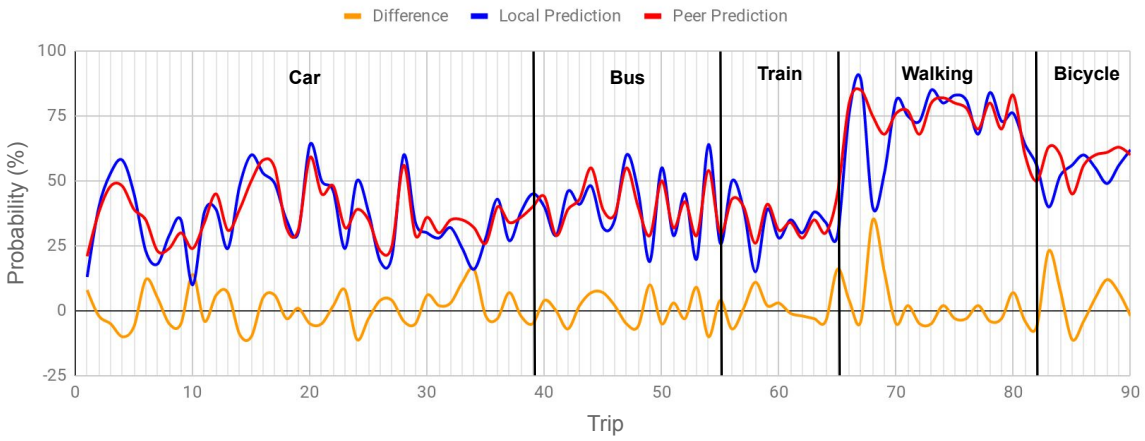

Figure 5.4: Comparison between local prediction and peer prediction.

In car trips, we observe that the peer prediction is higher than the local prediction in 19 (48.7%) occasions. If we only consider this prediction to take the decision, the peer predictions would correct the decision to car in 4 (10.3%) occasions (trip 6, 13, 33, 34) and change the decision to an incorrect mode in 1 (2.6%) occasion (trip 29). In bus trips, the predictions from the peers are higher than the local one in 18 (56.3%) occasions. The decision would change to the correct mode in 1 (3.1%) occasion (trip 45) while the correct decision is not changed in any scenario. In train trips, the peer predictions increase the probability in 5 (50%) occasions. The decision would be corrected in 1 (10%) occasion (trip 65). While walking, the predictions from the other devices are higher in 7 (41.2%) occasions and adjust the decision to the correct mode in 1 (5.9%) case (trip 68). In bicycle trips, the peer predictions are higher in 5 (62.3%) occasions and there was not any recorded event where the decision would change.

While the difference between the local and peer decision is not frequently higher than 15%, we can observe that when the local probability significantly drops, the peer decision offers a slightly better probability. This is relevant because when, due to the smartphone conditions, the local prediction is affected, the peer predictions can compensate with higher probabilities towards the correct mode.

The cases where the peer prediction is lower than the local one are not necessarily disadvantageous. For example, in a case where the local probability corresponds to 60% and the peer probability corresponds to 50%, we observe a 10% drop but the decision is still correct with a smaller confidence. We can slightly decrease the confidence of the correct predictions, but while offering an overall better percentage of correctly detected transports by compensating the low probabilities of the local decision.

5.4.2 Final mode decision

To evaluate the final decision in a P2P environment, we use the results obtained in phase 2 of the evaluation while considering the validations obtained in phase 1. Thus, each device already contains the validation history from its user. Table 5.2 shows the confusion matrix for userA which, as shown in Section 5.2, has a preference for the walk and car modes. Table 5.3 shows the confusion matrix for a different user (userB) that prefers the bus, walk and train modes.

	Predicted Mode				
Real Mode	Walking	Car	Bus	Train	Bicycle
Walking	100.0	0.0	0.0	0.0	0.0
Car	0.0	92.3	7.7	0.0	0.0
Bus	0.0	37.5	62.5	0.0	0.0
Train	0.0	10.0	0.0	91.0	0.0
Bicycle	11.1	0.0	0.0	0.0	88.9

Table 5.2: Final confusion matrix (in %) from userA

	Predicted Mode				
Real Mode	Walking	Car	Bus	Train	Bicycle
Walking	100.0	0.0	0.0	0.0	0.0
Car	0.0	74.4	23.0	2.6	0.0
Bus	0.0	6.2	87.5	6.2	0.0
Train	0.0	0.0	0.0	100.0	0.0
Bicycle	22.2	0.0	0.0	0.0	77.8

Table 5.3: Final confusion matrix (in %) from userB

	Predicted Mode					**Events**
Real Mode	Walking	Car	Bus	Train	Bicycle	
Walking	100.0	0.0	0.0	0.0	0.0	64
Car	0.0	80.1	16.0	3.9	0.0	156
Bus	0.0	14.0	84.4	1.6	0.0	64
Train	0.0	2.5	0.0	97.5	0.0	40
Bicycle	13.9	0.0	0.0	0.0	86.1	36

Table 5.4: Final confusion matrix (in %) obtained from the results of every user

Table 5.4 shows the confusion matrix that sums the final decisions obtained by every user. In this evaluation we considered 4 users travelling together for a total of 90 unique trips, specifically walking (16), car (39), bus (16), train (10) and bike (9). The results obtained from each user were aggregated to generate this matrix which means that each trip is considered four times (i.e., one evaluation from the perspective of each user).

Walking trips were detected with 100% accuracy. By analysing the modes used by each user, we observed that the walking mode is used with higher frequency when compared to the other modes, particularly with the bicycle which is the mode frequently confused with walking. Thus, when in doubt between bicycle or walking, the adjustment process increases the walking probability. Additionally, in the cases where the device can't accurately detect that the user is walking (i.e., user briefly stops, picks up the device from his pocket, etc.), the peers will still have strong walking predictions. These factors explain the high accuracy of our solution towards the walking mode.

While the adjustment process favours the walking mode, we still observe a satisfactory accuracy (86.1%) towards the bicycle mode. This is explained by the predictions from the surrounding peers,

which are strong enough (i.e., usually higher than 55%) to guide the decision in the direction of the bicycle.

Regarding car trips, we obtained a reasonable accuracy of 80.1%. The decision between car and bus highly depends on the user's preferences as the characteristics of these modes are quite similar, particularly in an urban environment where the travel speed is relatively low.

Similarly to the car, the results obtained for the bus vary according to the preferences of the user. In our experiments, we obtained an accuracy of 84.4%. However, only a small portion (6) of the bus trips consisted of repeated trips going through the same path of another trip found in the user validations. With more users validating trips over a higher amount of time, we believe that the bus accuracy would increase as the bus routes would be represented in the validations.

We can observe the relevance of the path score in train trips, with an obtained accuracy of 97.5%. The path represented by the train rails goes through multiple train stations. Thus, the path of one trip frequently consists in the union of smaller path sections separated by the train stations. Therefore, if one user validates a long train trip, it will help in the identification of all the train trips that go through smaller portions of that route. This effect is observed in our experiments.

Overall, we correctly classified 87.7% of the trips. It is important to note that a major part of the experiments was made in an urban environment. This conditions significantly impact the results obtained, particularly for car trips where the speed limit is quite low (i.e., maximum of 50 or 80 km/h) and there are multiple traffic lights in the trip, which results in stop times and speed metrics very similar to the bus trips. Given these conditions, the overall accuracy obtained is quite interesting.

Recall that DetectP2P performs the decision in near-real time without requiring a main server, and without access to online maps and transport schedules. This solution adapts to the users' preferences and takes into consideration the most frequent routes taken by the users. We believe that, with more validations from the users, the tendency would be to achieve higher accuracy values (i.e., more than 90%).

5.5 Power consumption

To evaluate the power consumption of our solution, we started a trip with 3 devices communicating with each other and registered the execution time (mins) for every 1% drop in the smartphone battery. We collected the consumption from a Xiaomi Mi A2 with a 3000mAh battery, while running only the DetectP2P application and with Bluetooth activated. Although, other services from the operative system were running in the background, which we can not control.

In Table 5.5, we show how the battery decreases during the test. We obtained an average of approximately 22 minutes of execution for every 1% of battery life. If we consider the minimum value obtained

Battery Drain (%)	1	2	3	4	5	6	7
Test Duration (min)	21	47	70	89	112	134	151

Table 5.5: Relation between the execution time (in minutes) and the smartphone battery (in %).

per battery drop, corresponding to 17 minutes, we obtain a battery life of approximately 28 hours. A smartphone with a battery capacity of 2000mAh would be able to run DetectP2P for approximately 18 hours, which is enough to collect trips during the average daily routine of a user.

5.6 Summary

In this chapter, we presented the evaluation of our solution relative to the requirements defined in Section 1.2. We split the evaluation into two phases. Firstly, we asked the users to individually collect and validate trips in their daily routine. In this phase, we were able to evaluate how DetectP2P adjusts the classifier decision to users' preferences and how the knowledge relative to the geographical routes grows with the increasing amount of validations. The execution time needed to calculate the path score was also evaluated in multiple scenarios. In phase two, the users were asked to travel together to evaluate DetectP2P in this scenario. The local decision of one user was compared with the decision obtained from the surrounding peers. Then, we showed the final results that evaluate the decision of our solution, in an environment where a group of users travel together. In our experiments, we correctly identified the modes of transport with an accuracy of 87.7% which is very reasonable considering that the tests were made in an urban environment with difficult conditions to fully distinguish the modes used. Finally, we tested the power consumption of DetectP2P running in a smartphone with a 3000mAh battery, obtaining an estimated battery life of 28 hours.

6

Conclusion

Contents

6.1 Conclusion

This work introduces a new technique to detect trips and identify the chosen mode of transport. We used a trip detection module that collects accelerometer and GPS data that is processed and then evaluated by a random forest classifier to predict a mode of transport. While the former solution runs in a stand-alone approach, we implemented a protocol to exchange information with the closest devices via Bluetooth. In our solution, each device is able to share its local decision with the surrounding instances of the application. Additionally, with a mechanism to validate trips, the devices can share validated trips with each other, obtaining relevant knowledge that can be used in their decisions.

To generate a decision, DetectP2P starts by collecting the prediction of the trip detection module and adjusting this initial prediction according to the validation history of the local user. This adjustment considers the performance of the classifier on the previous trips by analysing the false positives and false negatives obtained, while favouring the users' preferred modes. Then, a search goes through the local validations and the validations obtained from other devices in order to find trips that followed the route of the trip being evaluated and assign a probability value related with the frequency of each mode along that route. After considering the predictions received from peers during the trip, DetectP2P averages all the information to take its transport mode decision.

To evaluate this solution, a group of users helped by using the application in their daily routine. We started by comparing the initial results, consisting of the random forest decision, with the results obtained from the adjustment process. This evaluation simulated different adjustments based on different sets of validations so that we could see how the adjustment process evolved as the users validate trips. Then, we observed the tendency of the users to travel along repeated paths in their daily routines and showed how a validation from one user could help the future decision of others that go through a validated route. Finally, we evaluated the final prediction of this solution in a scenario where multiple users travel together and share information in real-time. In this scenario, DetectP2P was able to correctly identify the modes of transport with an accuracy of 87.7%.

6.2 Future Work

There is room for future improvements in DetectP2P. We imported the machine learning solution developed in Woorti. However, a different solution could be used to implement the Trip Detection module, particularly a solution that detects more transport modes.

Additionally, DetectP2P is limited to a maximum of 7 simultaneous connections due to the Bluetooth properties. By using a different communication link, such as Wifi-Direct, the number of connections could be increased. Although, the problems related to group negotiation would have to be addressed.

Bibliography

[1] G. Lugano, Y. Cornet, and A. Karadimce, "Worthwhile time in transport: Capturing the subjective value of the travel experience by smartphone," *ICT Innovations 2018, Web Proceedings ISSN 1857-7288*, pp. 205–222, 2018.

[2] (2019) The eu framework programme for research and innovation. [Online]. Available: https://web.archive.org/web/20191018132459/https://ec.europa.eu/programmes/horizon2020/

[3] (2019) Woorti. [Online]. Available: https://web.archive.org/web/20190514163554/http://www.woorti.com/

[4] R. Ross and J. Kelleher, "A comparative study of the effect of sensor noise on activity recognition models," in *Evolving Ambient Intelligence*, M. J. O'Grady, H. Vahdat-Nejad, K.-H. Wolf, M. Dragone, J. Ye, C. Röcker, and G. O'Hare, Eds. Cham: Springer International Publishing, 2013, pp. 151–162.

[5] I. Khan, S. Khusro, S. Ali, and J. Ahmad, "Sensors are power hungry: An investigation of smartphone sensors impact on battery power from lifelogging perspective," *Bahria University Journal of ICT*, vol. 9, pp. 8–19, 12 2016.

[6] X. Qi, Y. Li, M. Keally, Z. Ren, and G. Zhou, "Adasense: Adapting sampling rates for activity recognition in body sensor networks," 04 2013, pp. 163–172.

[7] J. Biancat, C. Brighenti, and A. Brighenti, "Review of transportation mode detection techniques," *ICST Transactions on Ambient Systems*, vol. 1, p. e7, 10 2014.

[8] A. Prelipcean, G. Gidófalvi, and Y. Susilo, "Transportation mode detection – an in-depth review of applicability and reliability," *Transport Reviews*, pp. 1–23, 10 2016.

[9] ——, "Transportation mode detection – an in-depth review of applicability and reliability," *Transport Reviews*, pp. 1–23, 10 2016.

[10] F. Biljecki, H. Ledoux, and P. van Oosterom, "Transportation mode-based segmentation and classification of movement trajectories," *International Journal of Geographical Information Science*,

vol. 27, no. 2, pp. 385–407, 2013. [Online]. Available: https://doi.org/10.1080/13658816.2012.692791

[11] A. Sauerländer-Biebl, E. Brockfeld, D. Suske, and E. Melde, "Evaluation of a transport mode detection using fuzzy rules," *Transportation Research Procedia*, vol. 25, pp. 591 – 602, 2017, world Conference on Transport Research - WCTR 2016 Shanghai. 10-15 July 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2352146517307512

[12] F. Biljecki, "Automatic segmentation and classification of movement trajectories for transportation modes," 07 2010.

[13] C. Zhou, H. Jia, J. Gao, L. Yang, F. Yixiong, and G. Tian, "Travel mode detection method based on big smartphone global positioning system tracking data," *Advances in Mechanical Engineering*, vol. 9, p. 168781401770813, 06 2017.

[14] C. Zavgorodnii, "DTBM - detecting travel and behavioral mode," Master's Thesis, Instituto Superior Técnico, Oct. 2018.

[15] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[16] Q. Han and D. Cho, "Characterizing the technological evolution of smartphones: Insights from performance benchmarks," in *Proceedings of the 18th Annual International Conference on Electronic Commerce: E-Commerce in Smart Connected World*, ser. ICEC '16. New York, NY, USA: ACM, 2016, pp. 32:1–32:8. [Online]. Available: http://doi.acm.org/10.1145/2971603.2971635

[17] B. Martin, V. Addona, J. Wolfson, G. Adomavicius, and Y. Fan, "Methods for real-time prediction of the mode of travel using smartphone-based gps and accelerometer data," *Sensors*, vol. 17, no. 9, 9 2017.

[18] J. C. Jeong and X. Chen, "Enhanced recursive feature elimination," in *2007 International Conference on Machine Learning and Applications(ICMLA)*, vol. 00, 12 2007, pp. 429–435. [Online]. Available: doi.ieeecomputersociety.org/10.1109/ICMLA.2007.35

[19] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava, "Using mobile phones to determine transportation modes," *ACM Trans. Sen. Netw.*, vol. 6, no. 2, pp. 13:1–13:27, Mar. 2010. [Online]. Available: http://doi.acm.org/10.1145/1689239.1689243

[20] I. Pál and A. Sauerländer-Biebl, "Automatische multimodale verkehrsmoduserkennung und situationserfassung mit hilfe von fuzzy-regeln," in *3. Interdisziplinärer Workshop Kognitive Systeme: Mensch, Teams, Systeme und Automaten*, 2014. [Online]. Available: https://elib.dlr.de/88823/

[21] H. Singh, M. Gupta, T. Meitzler, Z.-G. Hou, K. Garg, A. Solo, and L. A. Zadeh, "Real-life applications of fuzzy logic," *Advances in Fuzzy Systems*, vol. 2013, 08 2013.

[22] R. D. Das and S. Winter, "Detecting urban transport modes using a hybrid knowledge driven framework from gps trajectory," *International Journal of Geo-Information*, vol. 5, p. 207, 11 2016.

[23] H. Xia, Y. Qiao, J. Jian, and Y. Chang, "Using smart phone sensors to detect transportation modes," *Sensors (Basel, Switzerland)*, vol. 14, pp. 20 843–20 865, 11 2014.

[24] K.-Y. Chen, R. C. Shah, J. Huang, and L. Nachman, "Mago: Mode of transport inference using the hall-effect magnetic sensor and accelerometer," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 2, pp. 8:1–8:23, Jun. 2017. [Online]. Available: http://doi.acm.org/10.1145/3090054

[25] L. Stenneth, O. Wolfson, P. S. Yu, and B. Xu, "Transportation mode detection using mobile phones and gis information," in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS '11. New York, NY, USA: ACM, 2011, pp. 54–63. [Online]. Available: http://doi.acm.org/10.1145/2093973.2093982

[26] S. Hemminki, P. Nurmi, and S. Tarkoma, "Accelerometer-based transportation mode detection on smartphones," in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '13. New York, NY, USA: ACM, 2013, pp. 13:1–13:14. [Online]. Available: http://doi.acm.org/10.1145/2517351.2517367

[27] Y. Zheng, Y. Chen, Q. Li, X. Xie, and W.-Y. Ma, "Understanding transportation modes based on gps data for web applications," *ACM Trans. Web*, vol. 4, no. 1, pp. 1:1–1:36, Jan. 2010. [Online]. Available: http://doi.acm.org/10.1145/1658373.1658374

[28] A. Prelipcean, G. Gidófalvi, and Y. Susilo, "Mobility collector," *Journal of Location Based Services*, vol. 8, pp. 1–27, 10 2014.

[29] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," *SIGKDD Explor. Newsl.*, vol. 12, no. 2, pp. 74–82, Mar. 2011. [Online]. Available: http://doi.acm.org/10.1145/1964897.1964918

[30] M.-W. Lee, A. M. Khan, and T.-S. Kim, "A single tri-axial accelerometer-based real-time personal life log system capable of human activity recognition and exercise information generation," *Personal Ubiquitous Comput.*, vol. 15, no. 8, pp. 887–898, Dec. 2011. [Online]. Available: http://dx.doi.org/10.1007/s00779-011-0403-3

[31] L. Bao and S. S. Intille, "Activity recognition from user-annotated acceleration data," vol. 3001, 04 2004, pp. 1–17.

[32] B. SIG. (2019) Bluetooth. [Online]. Available: http://web.archive.org/web/20190915023548/https://www.bluetooth.com/

[33] M. Adarsh. (2019) Ble beacon technology made simple: A complete guide to bluetooth low energy beacons. [Online]. Available: https://web.archive.org/web/20190616175159/https://blog.beaconstac.com/2018/08/ble-made-simple-a-complete-guide-to-ble-bluetooth-beacons/

[34] K. Merry and P. Bettinger, "Smartphone gps accuracy study in an urban environment," *PLOS ONE*, vol. 14, no. 7, pp. 1–19, 07 2019. [Online]. Available: https://doi.org/10.1371/journal.pone.0219890

[35] M. A. Khan, W. Chérif, F. Filali, and H. Ridha, "Wi-fi direct research - current status and future perspectives," *Journal of Network and Computer Applications*, vol. 93, 06 2017.

[36] M. Collotta, G. Pau, T. Talty, and O. K. Tonguz, "Bluetooth 5: A concrete step forward toward the iot," *IEEE Communications Magazine*, vol. 56, no. 7, pp. 125–131, July 2018.

[37] A. Peshin. (2019) What is the range of bluetooth and how can it be extended? [Online]. Available: http://web.archive.org/web/20190717232105/https://www.scienceabc.com/innovation/what-is-the-range-of-bluetooth-and-how-can-it-be-extended.html

[38] R. Martin. (2019) Android multi bluetooth library. [Online]. Available: https://web.archive.org/web/20190228055824/http://arissa34.github.io/Android-Multi-Bluetooth-Library/