Pervasive and Mobile Computing 69 (2020) 101268

Contents lists available at ScienceDirect

Pervasive and Mobile Computing

journal homepage: www.elsevier.com/locate/pmc

edgeTrans - Edge transport mode detection

Paulo Ferreira [a,*], Constantin Zavgorodnii [b], Luís Veiga [b]

[a] University of Oslo, Postboks 1080 Blindern- 0316 Oslo, Norway
[b] Insitituto Superior Técnico/INESC-ID, University of Lisbon, R. Alves Redol 9, Lisboa, Portugal

ARTICLE INFO

Article history:
Received 31 January 2020
Received in revised form 29 August 2020
Accepted 1 October 2020
Available online 7 October 2020

Keywords:
Distributed systems
Mobile application
Transport mode detection

ABSTRACT

Automatic human activity detection, mainly transport-wise, is very relevant for architects and urban planners (among many others) when designing cities, roads, public transportation systems, etc. Such detection allows to better plan our cities and has been made possible with the widespread use of smartphones carrying several different sensors.

We developed edgeTrans, a system based on a smartphone application (app, for short), a database, and a server. The database stores the trips that were done, the server runs a machine learning algorithm that generates a model (i.e., a classifier) which is then integrated into the edgeTrans app. This app, after being installed, when running in a smartphone indicates the transport mode that is being used without requiring a network connection; it can now be downloaded from the Android Play Store or from the iOS iTunes (the app is called Woorti).

The results obtained in a real-world setting are very encouraging taking into account the requirements (e.g., accuracy, and low cost).
© 2020 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

Automatic human activity detection, mainly transport-wise, has been gaining interest not only because it allows to better planning our cities but also because new approaches have been made possible with the widespread use of smartphones carrying several different sensors [1–4].[1] In other words, the automatic detection of the transport used (instead of using paper-based surveys [6]) is of utmost relevance for architects and urban planners (among many others) when designing cities, roads, public transportation systems, etc. In addition, we believe that Transport Mode detection (TMD) at the edge will be important to develop next generation location based services (LBS). TMD will push the development of innovative and widespread LBS, by providing mechanisms to enable and foster the interaction (both monitoring and actuation) with the context surrounding people and things.

Most current solutions take advantage of smartphones to collect sensor readings, given the ubiquity of such devices, and use some machine learning (ML) algorithm constantly running in a powerful server (e.g., in the cloud) to detect which transport mode was used by each user [7]. The results obtained, even though better than those obtained otherwise (e.g., using surveys) are not exact, i.e., they are not fully accurate and are provided a long time after users have traveled; in fact, the collected data has to be sent to the cloud server, this server runs a ML algorithm to detect the transport mode used, and then a response is obtained. Such delay is even bigger if then the transport mode detected is sent to the

* Corresponding author.
E-mail addresses: paulofe@ifi.uio.no (P. Ferreira), constantin.zavgorodnii@tecnico.ulisboa.pt (C. Zavgorodnii), luis.veiga@inesc-id.pt (L. Veiga).

[1] In most developed countries, smartphone penetration reaches more than 80% of the population [5].

https://doi.org/10.1016/j.pmcj.2020.101268
1574-1192/© 2020 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

user's smartphone to allow her/him to correct the transport mode detected or if some award is to be collected by the user (in some shop for example). Thus, in other words, TMD at the edge gives us the opportunity for real-time detection (at a human scale) by omitting the network latency that transmission to a server (and back) imposes. Such real-time TMD allows context-awareness for LBS to provide customized information delivery based on a user's needs and her/his interactions. For example, the short delay allows a TMD application to alert a user when she/he should start walking towards the bus station in order to be on time for work [8], or to inform a user about the real time location of relevant buses [9]. Thus, solutions based on a server require constant network connection and, in addition, do not take advantage of the resources available in the smartphones.

The goal of edgeTrans is to detect the transport mode used by any person with a smartphone, while respecting the following fundamental requirements: (i) HAP — high accuracy regarding the transport mode detection for a large number of transports (namely walking, bicycle, car, bus, train)[2]; (ii) ED — ease of deployment, i.e., anyone with a smartphone should be able to deploy and use edgeTrans; (iii) MM — minimal maintenance, i.e., no particular operation should be required while edgeTrans is being used; (iv) UF — user-friendliness and minimal intrusiveness, i.e. the user should not be bothered; he just travels as he is used to; (v) LBC — low battery consumption, i.e., the user must note no particular difference on the battery usage of his smartphone when using edgeTrans when compared to a situation in which that is not the case; (vi) RES — responsiveness, i.e., the edgeTrans system must be quick enough (at human time scale) to detect the transport mode being used;[3] and (vii) CH — cheap, i.e., the cost of edgeTrans must be residual. Finally, we also consider the very relevant issue of privacy. In fact, any edge computing based solution clearly has such an advantage (when compared to a server-based solution) given that raw data needs not be sent to some server in the cloud. Transmitting raw data over the public Internet to a server in the cloud can incur into privacy risks [10] given that not only the user has a limited control over the data path from the smartphone to the server, but also because the smartphone may not have enough resources to encrypt/decrypt data. In addition, from a legal point of view, it is also important to acknowledge that sensitive data may be collected in one country and then transmitted to a server in another country with distinct legal regulations (e.g., the GDPR in Europe and the California Data Privacy Law in the US).

There are several challenges that we addressed regarding the design and development of edgeTrans. In particular, detecting a large number of transport modes with high precision while ensuring high-responsiveness and low battery usage of the edgeTrans application (app, for short). Obviously, we also had to ensure that, from a users point of view, the edgeTrans app implied ease of deployment, minimal maintenance, user-friendliness and minimal intrusiveness, and low cost. In this paper, we focus on the solutions related to requirements HAP, LBC, and RES.

edgeTrans is a system with a typical architecture, based on a smartphone application, a database, and a server. The app basically runs with no user intervention and collects sensor data (e.g., accelerometer) regarding the path that is being taken. This data is then opportunistically sent to the database where it is stored, and then accessed by anyone interested on such data. Such interest is currently mostly from data scientists (e.g., from eurecat — eurecat.org, as this is one of the partners from the EU MoTiV project — motivproject.eu) who access the data, after being anonymized, and obtain some statistics regarding the trips done. Note that there is an initial phase in which some trips are recorded and sent to the server for running a ML algorithm (based on Random Forest [11]). The only purpose for this initial phase is to train the ML algorithm and generate a model that is then integrated into the edgeTrans app. See in Section 5, how many trips were used for training.

The edgeTrans app can now be downloaded from the Android Play Store or from the iOS iTunes (the app is called Woorti). The app for Android is written in Java [12] and for iOS is written in Swift [13]. The edgeTrans server is written in JavaScript [14] using Node.js framework [15] and its web front-end in TypeScript [16] using the Angular framework [17]; the database that stores the users' trips is OrientDB [18,19].

The main contributions of this work can be summarized as follows: (i) a system that is being used in the real-world by a large number of non-computer knowledgeable users; (ii) an app that runs on the edge, i.e. on smartphones, and does not require network connection to detect the transport mode being used; (iii) a set of evaluation results that, contrary to many other solutions, are not lab-based but, instead, result from a real-world usage, and are being used by a large number of data scientists.

The paper is organized as follows. The next section addresses relevant related work. Sections 3 and 4 present the edgeTrans architecture and implementation, respectively. Section 5 shows some evaluation results, and the paper ends with some conclusions.

2. Related work

The need for information related with people's choices and behavior in the context of travels exists for decades. There are several areas [3,20] (transport, economy, health, energy, sociology, etc.) that are interested in this data and performed several scientific studies. By knowing a user's activity, a personalized service can be provided.

[2] Many existing systems simply consider walking, bicycle and motorized vehicles.

[3] Note that existing solutions that rely on a server (e.g., in the cloud) to run a ML algorithm and then provide the answer, do not respect this requirement.

A common difficulty with such studies is obtaining the data. As a matter of fact, the first studies were performed using traditional methods of data collection, including travel diaries, face-to-face or telephone or computer-assisted interviews [21]. Such traditional methods of data collection have some important limitations, namely the poor accuracy of the reported data, and the high number of non-reported trips due to respondent burden [22].

The appearance of new technologies, such as GPS, made it possible to perform passive studies, where data was collected using GPS loggers; with such loggers one can collect second-by-second location and position data without increasing the respondent burden [23]. However, the use of GPS loggers has its own limitations. The cost of GPS studies can be high and users can easily forget to charge their GPS loggers or forget them at home; thus, this results in an increased number of non-reported trips [24].

Smartphones allowed to reduce the impact of problems related to the use of GPS loggers, and made it easier to collect travelers data. The number of non-reported trips and costs associated with previous studies can be reduced when using smartphones to collect data; users tend not to forget their smartphones, and there is no need to purchase and distribute additional equipment [25].

2.1. Local vs remote TMD

In the last years several different systems were developed that support the collection of mobility data from smartphones. The problem of most systems is the impossibility of collecting the data in near real time (RES requirement). In fact, these systems [26–31] usually perform the detection of the transport mode used on the server side, while the smartphone is used only to collect raw sensor data. This means that the TMD is not only rather delayed with respect to the travel itself, but it also depends on the existence of network connection. In addition, given that it may be impossible to guarantee that there will always be network access, this may introduce even more delay in the TMD.

In short, most TMD systems are currently based on some ML algorithm and consist mainly on four steps: (i) data collection, (ii) pre-processing, (iii) feature extraction, and (iv) classification with a classifier obtained on a previous training phase of some ML algorithm [20]. **Remote activity detection** means that only steps (i) and (ii) are done in the smartphone; **local activity detection** means that most of the above steps (fundamentally step iv), are done locally, i.e. in the edge of the cloud (e.g., in a smartphone).

Thus, with remote TMD, after data is collected, it is gathered in some server (e.g., in the cloud) and a ML algorithm is executed to reach some conclusion with a certain confidence level (which, in current solutions is always less than 100%) such as *the user was in a car* or *the user was in a train*. Obviously, with remote data analysis, the activity detection is reached much later than when the activity takes place.

With local TMD, data is collected in the smartphone (where it may also be processed), and the detection of the transport mode is done locally. The ML algorithm used (more or less complex) may have had a previous training phase executed on the cloud, and it generates a classifier that runs in a smartphone. Thus, after being generated, a classifier (which must be adapted to resource-constrained devices such as smartphones with less resources when compared to a cloud server) runs in a smartphone and provides a response regarding the transport mode being used. This is done almost instantaneously, thus providing much better responsiveness (e.g., right after the end of a trip is detected).

Note that all solutions based on remote TMD fail to be an adequate solution given some of the requirements presented above; in particular, RES (responsiveness) given that all remote solutions do not ensure immediate responsiveness.

2.2. Local TMD

Some recent systems (e.g., Reddy et al. [9], Martin et al. [32], Sauerländer-Biebl et al. [33]) are local solutions that detect the transport mode in the smartphone where data is collected. All these solutions allow TMD in near real time and are very responsive, as the transport mode is provided almost instantaneously in the smartphone, support the detection of several different transportation modes, and used both the GPS and the accelerometer. The GPS sensor is rather energy demanding. For this reason, all these solutions require a large amount of power (in spite of some making efforts to minimize that consumption). Also note that detection accuracy depends on the transport mode being used; e.g., cycling detection in Reddy et al. [9] has less quality as the correct detection of such activity ranges from 45.6% to 87.9%.

Reddy et al. [9] uses an algorithm based on both a Decision Tree and the Discrete Hidden Markov Model. It combines data from the GPS and accelerometer sensors to classify between motorized transportation, walking, running, and biking. For this purpose, the authors use a decision tree followed by a first-order discrete hidden Markov model. Globally, it achieves an average of 93.6% of correct results (it can be as low as 45.6%) when tested on a data-set obtained from sixteen users. Clearly, this solution fails to fulfill the HAP, and LBC requirements. It is also not clear how the solution presented fulfills the ED, MM, UF, and CH requirements, as the authors do not address these issues. However, we assume that all such requirements are respected.

Martin et al. [32] uses an algorithm based on Random Forest. The solution, being computationally simple, can be executed on a smartphone. However, there is no indication if this system was really tested in a mobile environment and how much energy it consumes. It uses the Random Forest algorithm (their best classification model) which globally gives 96.8% correct results. It uses the smartphone GPS and accelerometer sensors data to classify a user's transportation mode. Five modes of transportation are detected: walking, biking, car, bus, and train. When compared to other previous

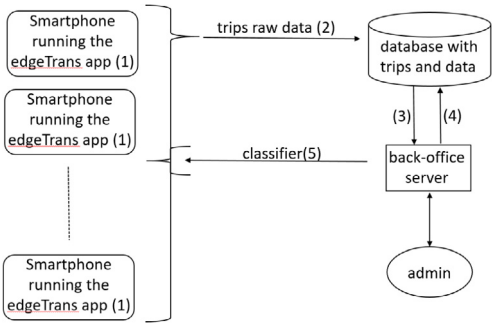


Fig. 1. High level global view of edgeTrans.

work (e.g., [34–40]), the authors say that their's is the first to develop models that include both GPS and accelerometer data while being applied in a real-setting, i.e., data is not collected under lab-controlled conditions. In fact, the data was collected by six students when participating in an undergraduate summer project in 2014. Clearly, the number of users (rather small) and being of a single-type (only students) makes the results obtained not widely applicable. The requirements HAP, ED, and LBC are not respected. It is also not clear how the solution presented fulfills the MM, and UF requirements, as the authors do not address these issues. However, we assume that MM, UF, and CH requirements are respected.

Sauerländer-Biebl et al. [33] uses a set of fuzzy rules applied to the data collected (using the GPS and the accelerometer). The authors are mostly interested in the detection of vehicular modes of transport; the overall precision obtained is 75%. There is no study regarding the battery usage. Thus, from the set of requirements listed in Section 1, no requirement is respected (with the exception of RES). However, we assume that both UF and CH requirements are respected as we see no reason for that not being the case.

A recent work is from Lu et al. [41]. It presents interesting results regarding the detection of transport modes. However, these results are obtained only when training the system, i.e., there is no real-world setting that is used for evaluating the solution provided. In addition, there is no information regarding where the system works (e.g., in a powerful server or on some other smaller device).

3. Architecture

This section presents the main aspects regarding the edgeTrans system starting with a global overview, showing all major components and their interactions, with a focus on the steps carried on. Then, each one of the most relevant steps is described in more detail: (i) collection of raw data, (ii) data processing, (iii) training phase, (iv) transport mode detection, and (v) dealing with lack of GPS. We then present some design options aimed at reducing battery consumption, and finally address the possible update of the classifier. Note that all the size parameter choices, selection of and detection thresholds mentioned in this section resulted from previous experimental evaluation. The evaluation of some of such options is presented in Section 5.

3.1. Global overview

Fig. 1 presents the main components and interactions of the edgeTrans system including the initial phase in which the training phase occurs. Note that once this initial phase ends, i.e., in normal usage, there is no need for such interaction between a smartphone and the server. edgeTrans app is downloaded from the Android Play store or the iOS iTunes, installed, and runs in each smartphone (step 1). While in the training phase, each smartphone sends to the edgeTrans database, users' trips raw data captured with its sensors via the edgeTrans app (step 2). Then, the back-office server reads the raw trips data stored in the server (file called *raw.csv*) and processes it (see more details later) so that a new file is generated; this new file (called *processed.csv*) has additional values which are calculated regarding averages, maximum and minimum accelerations and speeds during each trip which is also split in several segments (see precise definition later) (step 3). The *processed.csv* file is then accessed by the ML algorithm that, based on the data, and using a Random Forest algorithm, generates the corresponding model, i.e., a classifier (in a PMML[4] file [42]), that is the same for all users (step 4). From the PMML file, a binary version is generated so that the classifier can be incorporated into the edgeTrans app (step 5). Finally, the edgeTrans app, when running in a smartphone (i.e., in the normal and stationary case), based on the classifier that is part of the app (and is used by all users who have downloaded and installed the app) classifies each trip into one of the detectable transport modes, without requiring network connection (step 1). So,

[4] PMML stands for Predictive Model Markup Language. More details in Section 4.

there is a first phase (collection and training of the ML algorithm) including the steps mentioned above; then, starting with step 5 (that closes the cycle), users are free to use the edgeTrans app to detect the transport mode used. Clearly, the whole cycle described above, keeps on going, with trips being sent to the database, and new versions of the classifier being generated and uploaded to the edgeTrans app, thus increasing the quality of the detection of the transport mode used. The frequency of such updates is out of the scope. Note that such a new version resulting from a classifier update is not required for the system to keep on working. In addition, trips are sent to the database only if there is some other reason for the user to connect to the network. In fact, as already mentioned, the system keeps on working, detecting trips and transport modes with no need for network connection.

Thus, in short, the several steps that are to be considered, are the following: collection of raw data, processing of the collected data, training phase, and transport mode detection (done in the smartphone). Note that, to identify the transport mode in near real time, a smartphone is capable of detecting the transport mode based on the classifier integrated into the edgeTrans app without having to access the network. Each one of these steps mentioned above is described below in more detail.

3.2. Collection of raw data

The collection of raw data, captured by the sensors in a smartphone, is done by a smartphone running the edgeTrans app, and then is sent to the database server. The above mentioned data is obtained from the sensors accelerometer and GPS. It is well know that GPS uses a lot of energy; so, to minimize such consumption, we use the accelerometer to detect that movement has started and only then turn on the GPS sensor (i.e., the user does absolutely nothing).

3.2.1. Time and distance configurations

The edgeTrans app allows both automatic and manual (collection) through the corresponding interface. In either case, at the end of a trip, a user has the option of validating the result, i.e., the transport modes used during the trip.

Based on our initial experiments with edgeTrans: (i) GPS tracking is turned on when the edgeTrans app detects a 10 s interval with average accelerations above 3 m/s^2, or once every 3 min (the one that occurs the earliest); (ii) a new trip is detected as having started when the edgeTrans app detects a displacement of at least 100 m over a 5 min interval; and (iii) a trip ends when the distance traversed in the last half-hour is less than 100 m.

Another relevant aspect is that, while a trip is going on, it is natural that a user changes from one transport mode to another, and this usually implies some waiting time. Note that such waiting time is part of the trip being done and should not be confused with the starting of another trip. Thus, we have introduced the notion of a waiting event. This is defined as a period of time, in a single trip, during which the user, e.g., may change from one transport mode to another. Once again, based on our initial experiments, edgeTrans automatically detects a waiting event when a user takes at least 5 min without traversing a distance longer than 100 m.

During a trip, the raw data of the GPS and accelerometer sensors is collected and saved in the internal memory of a smartphone. The accelerometer data is collected once per second and the GPS data is collected every ten seconds. These intervals (1 and 10 s) were selected after several initial tests, in order to minimize the battery consumption.

Note that all the above values for time and distance were decided after an initial testing phase of edgeTrans. Then, the app was configured accordingly and the results presented in Section 5 on evaluation assume their usage.

3.2.2. Data collected

When the edgeTrans app is being used in the normal case (i.e., after the training phase has been finished and the classifier generated) a trip can be sent to the database after being validated, if and when a network connection is available; note that for the detection of a transport mode, there is no need for such network connection. The database stores all the raw data collected by each smartphone along with the automatic transport mode that was detected and with any corrected (by the user) data; this data can then be freely used by anyone interested (urban planners, data scientists, etc.).

In the initial version of edgeTrans, i.e., while the system is being trained (see more details in Section 3.4) the raw data that was collected included 537 trips, collected by 15 users, and its total duration (of the collected data) was approximately 265 h, being split into different transport modes as follows: walking (62.07 h), bicycle (15.53 h), car (69.68 h), bus (65.82 h), and train (49.92 h). In average, each trip included 2.1 transport modes. For training and testing the ML algorithm, it was used 75% and 25% of such raw data, respectively.

3.3. Data processing

In the training phase, several input parameters (called features) for the ML algorithm were obtained after processing the raw data that was collected by the edgeTrans app and stored in the database. After a manual verification and filtering of the reported data (e.g., some of the trips were wrongly reported by the user) in addition to automatic verification and filtering, this processing of the raw data is done in 2 steps: (i) division of the filtered data into segments of the same size; (ii) feature extraction from each segment. Section 5 shows the performance results of edgeTrans taking into account a classifier that was generated according to these steps.

3.3.1. Data segmentation

Both when being used in the normal case (i.e., after the training phase has been finished and the classifier generated) and also during the training phase, the task of detecting a transport mode is complicated by the fact that users may use more than one transport mode in each trip. Thus, the first thing that is done, is finding the number of transport modes used during each trip.

In a single trip, to detect when the transport mode changes, and taking into account the notion of a waiting event, edgeTrans splits the raw data into segments of fixed duration and uses the classifier on each one of these segments. In this way, it is possible to detect when a sequence of segments with the same transport mode is followed by another sequence with a different transport mode; this allows to identify when a change of the transport mode occurs.

To better detect when the transport mode changes, some overlap between segments has also been applied. This means that part of the raw data of one segment is also used as a part of the next segment.

Segment data division is performed by calculating the time between two sensor readings. This calculation is possible because each raw data item contains a timestamp (reflecting the moment when it was captured); that item is stored in the server in that order when the trip is later on sent (opportunistically) from the smartphone. When the time interval between two timestamps is equal to the size of the segment, this means that all the data collected between these two readings belongs to the same segment.

For each segment size (measured in seconds), the corresponding overlap sizes (also measured in seconds) are as follows: (i) segment size = 30 − 0, 10 and 15; (ii) segment size = 60 − 0, 20 and 30; (iii) segment size = 90 − 0, 30 and 45; and (iv) segment size = 120 − 0, 40 and 120. The sizes were chosen as a result from our experience with preliminary versions of the edgeTrans system.

3.3.2. Feature extraction

Regarding the training phase, from the raw data of each segment, edgeTrans calculates a set of features; these features describe the corresponding segment. The resulting features are then used to train the ML algorithm.

In total, there are 23 features extracted from the raw data of each segment. The extracted features are the following: `avgAccel, maxAccel, minAccel, stdDevAccel` — average, maximum, minimum and standard deviation of acceleration (units in m/s^2); `avgFilteredAccel` — average of acceleration magnitude after removing the values below acceleration filter (units in m/s^2); `accelsBelowFilter` — percentage of accelerations with values below the acceleration filter; `accelBetw03-06` — percentage of accelerations with values between 0.3 m/s^2 and 0.6 m/s^2; `accelBetw-06-1` — percentage of accelerations with values between 0.6 m/s^2 and 1.0 m/s^2; `accelBetw-1-3` — percentage of accelerations with values between 1.0 m/s^2 and 3.0 m/s^2; `accelBetw-3-6` — percentage of accelerations with values between 3.0 m/s^2 and 6.0 m/s^2; `accelAbove-6` — percentage of accelerations with values above 6.0 m/s^2; `avgSpeed, maxSpeed, minSpeed, stdDevSpeed` — average, maximum, minimum and standard deviation of speeds (units in km/h); `avgAcc, maxAcc, minAcc, stdDevAcc` — average, maximum, minimum and standard deviation of precision associated with location coordinates; `gpsTimeMean` — mean time between 2 consecutive locations received (units in seconds), `distance` — distance traveled during the segment (units in meters); `OS` — operating system of the device where the raw data was collected (Android or iOS); and `estimatedSpeed` — value that represents if the speed related features were calculated using the location of the segment or were estimated using the locations of the neighboring segments.

The intervals defined for accelerations represented by the features named `accelsBelowFilter, accelBetw-03-06`, and `accelBetw-06-1`, were defined after analyzing the acceleration data related to several transport modes. It was observed that the transport modes car, bus and train tend to have low accelerations (rarely exceeding 1 m/s^2). Therefore, in order to facilitate the distinction between these modes, acceleration values between 0 and 1 m/s^2 were divided into 3 categories of different intervals.

In addition, the value chosen for the `accelsBelowFilter` feature was 0.3 m/s^2, because in the analyzed trips this value represents a border below which it is possible to consider that a user is staying still, thus not moving.

It was also observed that the accelerations associated with bicycles are usually below 6 m/s^2, while the accelerations associated with walking may exceed this value. For this reason, it was considered that 6 m/s^2 is a useful value to define a boundary between intervals (resulting in the feature `accelAbove-6` mentioned above). Finally, it was also decided to split the interval between 1 m/s^2 and 6 m/s^2 in two to obtain a better separation of accelerations in this interval. As a result, features `accelBetw-1-3` and `accelBetw-3-6` were introduced.

3.4. Training phase

The training phase of the ML classifier is based on the features calculated for each segment during the data processing phase. During the training phase it was necessary to test several alternatives of classifier configurations in order to find the configuration that allows to achieve the best results. It is considered that a classifier configuration is defined by: (i) the ML algorithm used, (ii) the hyperparameters of the algorithm used, (iii) the features of the model, and (iv) the segment and overlap sizes.

The ML algorithms that were used when developing edgeTrans were the Decision Tree and the Random Forest [11]. The identification of the transport mode with these two algorithms does not require very demanding computation. This property is very important considering that the detection of the transport mode occurs on a resource constrained device as a smartphone (when compared to a server on the cloud, it has less computer power, it is battery limited, etc.).

As already mentioned, we opted for the Random Forest algorithm based on the results obtained (more details in Section 5).

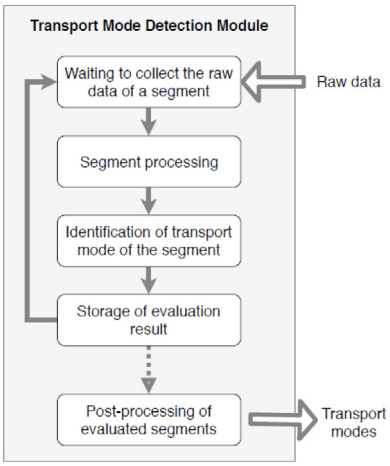


Fig. 2. Architecture of the TMDM (TransportMode Detection Module) in edgeTrans.

3.5. Transport mode detection

The classifier resulting from the ML training phase was used in the Transport Mode Detection Module (TMDM) of edgeTrans to detect the transport modes used during a trip. Note that the TMDM runs on smartphones (Android and iOS) when the user is on a trip. When the trip ends, the TMDM returns a list with all the transport modes used.

Fig. 2 represents the sequence of steps performed by the TMDM during a trip identification (all these steps are done in a smartphone). The raw sensor data (location and accelerometer readings) are the input of TMDM. This data is received in real time during a trip. The TMDM waits until the raw data corresponding for one segment is collected. The identification of the data belonging to a specific segment is then performed by calculating the time passed between two readings of the sensors, in the same way as it was done during the data segmentation phase when the classifier was trained (as described in Section 3.3.1). After the raw data corresponding to one segment is collected, this segment is processed. Since the classifier evaluates the same features used to train the classifier, the segment data is processed using the same processing steps that were applied during the feature extraction phase when the classifier was trained (as described in Section 3.3). After processing the raw segment data, the extracted features are evaluated in order to identify the transport mode. This step is described in more detail in the next section (Section 3.5.1). The evaluation result is then stored in the TMDM.

The result consists in a list with probabilities for each transport mode. After processing a segment, the TMDM waits for more raw data (which will be part of the next segment). This cycle is repeated until the end of the trip. When the trip ends, the post-processing phase is performed. All the results stored for each one of the trip segments are processed in order to obtain the list of transport modes used during the trip, and the timestamps when the transport mode changes occurred. This list represents the final result of the TMDM. The post-processing phase is described in more detail in Section 3.5.2.

The TMDM assumes the existence of an upper layer responsible for explicitly indicating that a trip is started or ended (following the rules mentioned above in Section 3.2.1), i.e., the TMDM does not include the automatic trip start and trip end detection. When a trip begins, the raw sensor data is sent to the TMDM. Then, the trip ends and the TMDM is informed about that, to perform the post-processing phase. After that, the TMDM is ready to identify the modes of transport of the next trip.

Opportunistically, i.e., for some reason, when the user connects the smartphone to the Internet, all the trip data is sent to the edgeTrans database.

3.5.1. Transport modes

The classifier that runs in each edgeTrans app (obtained during the training phase) allows to detect the transport modes in which there is movement (walking, car, train, etc.). However, users frequently stop for short periods of time during a trip (e.g., at traffic lights, to buy a newspaper, etc.). Thus, if a user stops for a period of time, the corresponding segments of raw data will be classified accordingly to the five modes of transport known by the classifier (walking, bicycle, car, bus, and train).

Note that, if the classifier was not able to detect the still mode, these segments would be mistakenly identified with another most probable transport mode. The more segments are poorly identified, the more difficult it would be to identify correctly the transport modes used in a trip. Therefore, to decrease the amount of noise it is absolutely necessary to detect the still mode. For example, from the point of view of a user, when he is driving, being stopped at a traffic light makes part of riding a car; so, on his mind, he only used one transport mode (car).

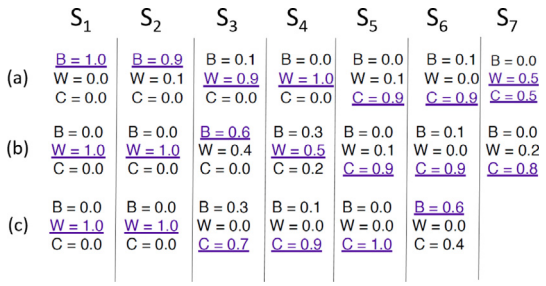


Fig. 3. Examples of: (a) strong segments detection; (b) merging the sub-sequence of strong segments with candidate sub-sequence; and (c) most probable transport mode for the sub-sequence [S3; S6].

The classifier can easily confuse the still mode with other modes of transport. For example, a train or a car have low accelerations, as it happens with the still mode. In addition, when the location information does not exist or is not fully accurate, the classifier may easily fail to identify the correct transport mode. However, it is relatively easy to set the values of the still mode boundaries because standing means that both speed and acceleration are low.

Thus, the still mode is detected by checking the values of two features (presented in Section 3.3.2): `avgSpeed` and `accelsBelowFilter`. The threshold values for these features (determined when testing a preliminary version of edgeTrans) are the following: `avgSpeed` is less than or equal to 2.5 km/h, and `accelsBelowFilter` is greater than or equal to 0.8 km/h. When the above features respect these thresholds, the user is considered to be still. Thus, the transport mode of the corresponding segment is still.

Otherwise, i.e., if the transport mode is not still, the segment is evaluated by the classifier according to the five transport modes known by the classifier. Thus, we can say that, in total, the TMDM allows to identify six modes of transport (walking, bicycle, car, bus, train, and still).

3.5.2. Post-processing phase

The classifier is not perfect, i.e., some segments may be tagged with the wrong transport mode. This means that if the transport modes detected for a sequence of segments of a trip were immediately used, the trip would have many more modes of transport detected than those that were actually used. To avoid this erroneous detection and filter the noisy segments, the sequence of results of the segments must be processed before creating the final result. The post-processing of segments is done in 6 steps, as follows.

(1) Detection of sub-sequences of strong segments. A strong segment is defined as a segment whose most probable transport mode is walking (i.e., the probability is at least 0.8). The main idea is to divide the total sequence of segments into sub-sequences using the walking periods as a separator. In fact, usually a user needs to walk, even for a short time interval, in order to change his mode of transport. Walking is characterized by high acceleration and low speed. In general, the classifier can easily identify this mode (more details in Section 5). Thus, edgeTrans splits the entire sequence of a segment into sub-sequences using strong segments as separators; this allows edgeTrans to detect the precise number of the transport modes that were used. This idea is represented in Fig. 3(a); it shows a possible sequence of segments, from S1 to S7, and the strong sub-sequences identified after applying this step (S3 and S4). For each segment, it is also presented the probabilities for the following transport modes: bus (B), walking (W), and car (C); the highest probability is underlined.

(2) It may happen that, between two strong segment sub-sequences, there are other segments sub-sequences whose transport mode is most likely to be walking, although the probability associated is not sufficient to consider them strong segments. These sub-sequences are defined as candidate sub-sequences. As there are strong sub-sequences around candidate sub-sequences, it makes sense to consider these candidate sub-sequences as part of the strong sub-sequences if they are separated only by few segments with another transport mode. (More details on the number of such separating segments below.) Thus, separation segments are defined as segments that are between a strong sub-sequence and a candidate sub-sequence.

The decision on merging the strong sub-sequence and candidate sub-sequence depends on the size of the candidate sub-sequence ($|C|$) and the number of separation segments ($|S|$). The merge occurs when it is possible to satisfy the condition $|S| \leq (|C| + 2)/2$. If the merge occurs, the strong sub-sequence will include the separating segments, and the segments of the candidate sub-sequence. Note that this merge condition was found experimentally while developing the preliminary versions of edgeTrans.

For example, Fig. 3(b) represents a sequence of segments which includes a sub-sequence of strong segments (S1 and S2), a sub-sequence of candidate segments (S4), and a separation segment (S3). Since the merge condition is satisfied, the merge occurs, and the sub-sequence of strong segments is now defined by the segments S1, S2, S3, and S4.

(3) In addition to walking, the still mode also works as a separator between two other modes. Thus, for sub-sequences that are between two sub-sequences of strong segments (or between the beginning and end of the considered sequence),

it is applied a similar idea as described in step 1; however, instead of walking, edgeTrans uses periods of still to separate the sub-sequences.

(4) Knowing the sub-sequences of strong segments and still sub-sequences, allows edgeTrans to determine when a user was walking or was still. The remaining transport modes are then identified. This is done by analyzing the sub-sequences of segments that are between two known sub-sequences (from step 1 or step 3) or between the beginning/end of a trip. The probabilities of each transport mode are summed and the transport mode with the highest probability is considered to be the mode of transportation of these segments.

The idea underlying this step is illustrated in Fig. 3(c). The represented sequence of segments contains a sub-sequence of strong segments at the beginning ([S1; S2]). The transport mode is identified for the remaining sub-sequence ([S3; S6]). The transport mode identified is considered to be car because the sum of probabilities is higher for this mode ($0.7 + 0.9 + 1.0 + 0.4 = 3.0$).

(5) When this step is reached, the segments sequence being considered is already completely split into sub-sequences associated with a transport mode. However, some of these sub-sequences represent very small (human-scale) time intervals (from seconds to few minutes), meaning that these sub-sequences are very likely just noise; that is the case if the transport mode identified for these sub-sequences is neither walking, nor still. Taking this into account, some simple rules were created to check the neighboring sub-sequences, and join very small sub-sequences to larger sub-sequences.

(6) Finally, all segments sub-sequences of the identified transport modes are analyzed to join the consecutive sub-sequences with the same transport mode. The resulting list of segments sub-sequences is the final result of the TMDM.

3.6. Dealing with the lack of GPS signal

Under normal usage (i.e., after the training phase is finished and the classifier generated) in order to determine the mode of transport independently of the smartphone orientation, for each accelerometer reading, edgeTrans calculates the corresponding magnitude (M) using the formula $M = \sqrt{x^2 + y^2 + z^2}$, where x, y and z represent the acceleration along the corresponding axis. In addition, as mentioned above, every 10 s, the GPS location is registered.

However, one limitation of the GPS sensor is the loss of signal in certain locations such as tunnels, underground spaces or indoors. Thus, without such location information, it becomes impossible to calculate the features mentioned above (in Section 3.3.2). Therefore, there is a loss of useful information that helps to identify the transport mode.

To reduce the impact of the above mentioned GPS signal loss, edgeTrans makes a speed estimation that works as follows. When a segment without location data is found (thus, corresponding to a situation in which there is no GPS signal), that segment is inserted into a waiting list. When a following segment, with location data, is detected (corresponding to a situation in which the GPS signal is back), the average speed is calculated for all the segments in the waiting list. Thus, the average speed is calculated as a whole for all the segments including the one that was found lately, and those in the waiting list.

For example, consider the following possible sequence of segments: …, $Sv1$, $Sn1$, $Sn2$, …, Snk, $Sv2$, …where Svi represents a segment with the corresponding data location, and Sni represents a segment without location information (e.g., given that the corresponding smartphone entered into a tunnel and there is no GPS signal). Thus, all the segments between Sn1 and Snk are inserted into the waiting list. When segment Sv2 is detected, it becomes possible to estimate the speed for segments in the waiting list using locations from segments Sv1 and Sv2. After the speed estimation, all the segments are removed from the waiting list. It is important to note that when the sequence of segments ends or begins with segments without locations, it becomes impossible to estimate the speed. In this case all the speed-related parameters are initialized to zero.

3.7. Battery consumption

To be effectively adopted by users, we designed the edgeTrans app while trying to minimize its use of the battery. Otherwise, users may either not adhere to the app (i.e., not install it altogether) or give up using it after a while. Therefore, in order to limit battery consumption, we adopted a number of optimization strategies that follow a staged approach, adapted to the specifics of each operating system targeted (Android, iOS). The basic idea is the following: sensors providing more relevant information, but also consuming more power, are engaged only when they are deemed necessary and relevant for the purpose of the application. This means that: (i) we delay the GPS activation, and (ii) use an adaptable sample frequency for the sensor readings. Such design options (explained below) allow the edgeTrans app to have low energy consumption as presented in Section 5.

3.7.1. Motion-triggered GPS activation

We delay using GPS activation for trip detection, i.e., we delay request GPS coordinates, irrespective of it being turned on, until the application detects relevant motion. Thus, when the app starts and when the user is stationary (i.e., no relevant motion detected), no GPS requests are performed. This way, stand-by GPS uses little power when compared to GPS actively seeking signals and processing them in order to provide coordinates to the edgeTrans app.

Considering relevant motion detection, in Android and iOS the detection that a user is moving is carried out differently; this is due to the way Android and iOS deliver location, acceleration and motion information. In iOS, to detect a user

is moving when the GPS is off, we use the standard Activity detection from iOS. This activity detection notifies the edgeTrans app when the user has started an activity involving motion (i.e., walking). Additionally, the Activity detection notifies the app when there is no motion-related activity, but some residual acceleration was detected (e.g., stationary), in a loosely periodic fashion (every 4 or 5 s). These events are related with motion with very soft accelerations but that may accumulate over time across a significant distance (typically when traveling by train where for large distances acceleration may always be very soft). Therefore, edgeTrans app piggybacks on these events and periodically (roughly about 3 min) tracks movement with a GPS request and assesses whether the distance covered in the last 5 min is over the 100 meters threshold, regardless of acceleration readings.

In Android, we use raw accelerometer data to check if there is likelihood that a user is moving, by computing the moving average acceleration for each ten seconds period (checking whether it is over 2.5 m/s^2 — similar to walking). Additionally, there are situations where long periods of time, with very soft acceleration, may accumulate across large distances (typically, train); to account for such situations, we track movement with a GPS request periodically every 3 min (and then checking whether the distance covered is over the 100 meters threshold).

In summary, these strategies achieve the following: (i) they enable the edgeTrans app, when the user is stationary, to reduce the GPS sampling rate to 1 request over 3 min (180 s), which is very low and at least one order of magnitude lower than during travel; and (ii) they still allow the edgeTrans app to detect relevant movement and start detecting trips, even if the threshold distance covered results from the accumulation of very soft movements across a period of time (e.g., as typically in a train).

3.7.2. Sampling frequency and GPS deactivation

The accelerometer data is collected once per second and the GPS data, during travel, is collected every 10 s. These frequencies were selected also in order to minimize battery consumption during travel, while ensuring enough accuracy regarding the travel path.

When it is determined that there is no relevant movement, in iOS the time until we turn off the GPS is 1 min. This is done to prevent a high use of battery when not in travel, suitable for iOS due to the GPS having a relatively higher accuracy rate, when compared to Android.

In Android, where accuracy can be significantly lower during the initial GPS requests (in several models), the time to turn off the GPS is 5 min to reduce the probability of a false negative happening (missing a starting trip).

3.8. Updating and specializing the classifier

As already mentioned in the beginning of this section, a classifier can be updated as training is reiterated including more and/or newer trips. In particular, this may happen given that new results (trips recorded) can be uploaded opportunistically to the server (by users, knowingly and willingly) and a new training phase can have such data into account (naturally, it should only include trips from trusted users so as to avoid feeding wrong data into the classifier during re-training). The resulting new or updated classifier can then be integrated into a new version of the edgeTrans app and become available and fully disseminated, in a compatible and incremental fashion, as users update their edgeTrans app installed locally in their devices (at the edge).

Furthermore, such a re-training phase, and the corresponding resulting classifier, may be unfolded into multiple customized versions. For instance, each variant could result from choosing from the following broad spectrum of approaches of trips to consider in the re-training process: (i) include all trips of the initial training and, additionally, a number of trips specific to each city for customization, (ii) include a mix of trips from the initial training and trips specific to each city for customization, increasing the percentage of city-specific trips in the training set as more become available, and (iii) after sufficient city-specific trips are obtained, include only city-specific trips in the re-training set. Note that this entails a relevant number of trusted users supporting the app in each city, and enough trips submitted by them, in order for each customized classifier to be effective.

Additionally, customization to specific cities raises an architectural decision regarding the application: either (i) include the code and model data of all classifiers (e.g., the general one and all those customized and tuned for each city) in each edgeTrans app version, (ii) deploy different variants of the edgeTrans app (one generic and several city-specific), or (iii) follow an hybrid of the above two approaches and deploy a single variant of the app with the generic classifier, and make each city-specific classifier code and model as downloadable content. Option (i) incurs into additional storage overhead across all users' devices with the classifier code and model for all cities, while maintaining uniformity. Option (ii) avoids storage overhead but in effect requires the deployment of a specific edgeTrans app customized for each individual city with added overhead when managing application updates. Option (iii) combines avoiding storage and update management overhead at the expense of a more complex application architecture supporting downloadable content (code and model) for each city-specific classifier.

Moreover, customization needs not be limited to geography. Other criteria can be taken into account (provided there are enough trusted users and enough submitted trips), such as existing and/or used transports in each city, seasonal transports used in each city across the year (and also regarding weekday/weekends, day-time/nigh-time patterns) and even individual personal behavior and preferences. We believe that such an evolution will allow future versions of edgeTrans app to have even better accuracy (see evaluation results in Section 5) for each individual user, and is now made possible to be implemented in the future. Such improvements are ongoing and future work and out of scope of this article.

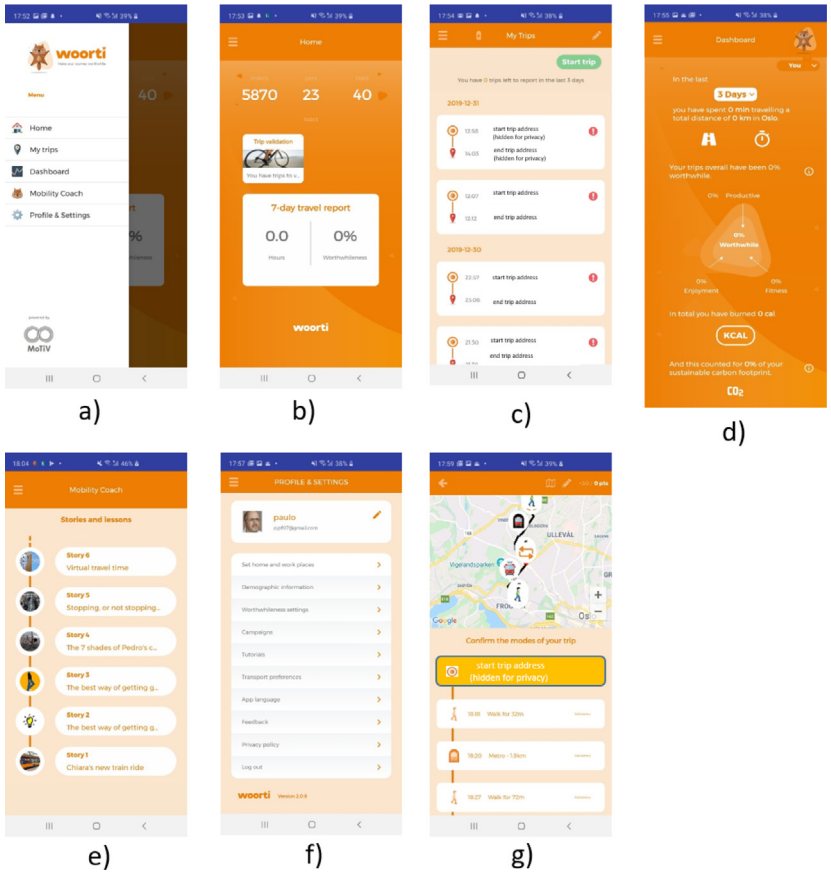


Fig. 4. Screenshots of the application: (a) `main`, (b) `home`, (c) `my trips` (d) `dashboard`, (e) `mobility coach`, (f) `profile settings`, and (g) `my trips in Oslo`.

4. Implementation

In this section we present some aspects of the implementation (the software used to develop the system), and show some screenshots of the application while running in a smartphone (Samsung S1 with Android 10).

4.1. Software used

As already mentioned, we tested the preliminary versions of edgeTrans with two ML algorithms, Decision Tree and Random Forest. Based on the results obtained we opted for the later (in the following, we refer only to it).

The classifier was trained in Python, using the ML algorithms defined in scikit-learn library [43]. To be used on Android and iOS, the resulting classifier was exported using 2 different formats, one for each operating system.

To be used in the edgeTrans app running on Android, the classifier was exported to the PMML format [42] using the JPMML library. The JPMML library allows to export the model to PMML format and to execute this model in Java that is then used in Android. To be used in the edgeTrans app running on iOS, the classifier was exported to the Core ML model format. The Core ML framework allows to export and use the ML models on iOS devices. With JPMML and Core ML, it was possible to create the ML model in scikit-learn and then use that model on the Android and iOS operating systems. The backoffice server is implemented in JavaScript with Node.js with the library Express. This combination allowed us to focus on the web development making it to work on a short period of time. The database, OrientDB, is graph based well adapted to store trips information. Java PMML, Core ML Framework, Node.js, Express, and OrientDB are all available from the corresponding websites.[5]

[5] Java PMML: openscoring.io; Core ML Framework: developer.apple.com/documentation/coreml; Node.js: nodejs.org; Express: expressjs.com; OrientDB: orientdb.com. All acccessed in 30-01-2020.

4.2. Application screenshots

The most relevant screens of the edgeTrans app are show in Fig. 4: (a) `main` screen, as the name suggests, is the entry screen of the app from which the user can go to any other screen (`home` screen in particular which is partially hidden); (b) `home` is the screen that presents a summary of some aspects of the users trips; (c) `my trips` shows a (scrollable) list of the trips done so far by the user; (d) `dashboard` shows a screen with some indicators regarding the trips done so far; the picture is longer (than the others) as its final part (at the bottom) is only seen once the user scrolls down; (e) `mobility coach` is a screen that shows some stories, lessons learned, suggestions, etc. regarding the trips done by the user; (f) `profile and settings` is the screen that allows the user to define some aspects of the app (e.g., the language used which can be one of twelve); and (g) `my trips in Oslo` for a particular trip once the user picks one among the several shown (screen c).

An important aspect to consider is that the user can correct the transport mode that was automatically detected by the app; this can be done from the screen shown in Fig. 4(g) allowing the correction of any aspect of the trip done. We do not show such screens for lack of space.

5. Evaluation

The evaluation presented in this section is divided in two main parts. The first shows the results obtained with the two ML algorithms that were used in the preliminary edgeTrans versions: Decision Tree and Random Forest (as already mentioned, we then picked the Random Forest algorithm). Then, the second part of this section (Section 5.2) regards the Random Forest algorithm while being used in a real-world setting where the edgeTrans app is being used by a large number of users.

5.1. Initial edgeTrans versions

This section addresses the most relevant issues regarding the evaluation of the edgeTrans while being used for training purposes and to generate the classifier. The evaluation process consists in three steps: (i) Division of all the collected trips into training and testing sets. About 25% were used for the test set and the remaining 75% were used for the training set. (ii) Training and evaluation of different classifier configurations using the training set. The evaluation results obtained and the comparison of different configurations are described next. (iii) TMDM evaluation using the test set; this is described in more detail in Section 5.1.2. The metrics used to perform the evaluation of the classifier were: confusion matrix, precision, recall and F-score. More details about these metrics can be found elsewhere [44].

Each one of the two ML algorithms has specific parameters (hyper-parameters) related to its internal operation. The main difference between the hyper-parameters and the features (or parameters) of the model is that the hyper-parameters do not depend on the problem to be solved.

To find the best classifier, we exercised several combinations of classifier configurations. The results using different algorithms, segment sizes, overlap sizes and values of hyper-parameters were compared in order to find the best classifier. For each feature, we evaluated the corresponding importance on the correct transport mode identification. The importance was calculated using the RFE algorithm (Recursive Feature Elimination algorithm) [45] and the features that worsened the results of the classifier were ignored.

5.1.1. Classifier

During the training phase (described in Section 3.4) different configurations of the classifier were tested. The goal was to find the configuration that allows to achieve the best results. Thus, for each combination of the algorithm, segment size, and overlap size, the best hyper-parameters values were selected along the best set of features. The best result found for each size of segment and overlap, according to the algorithm being used, is presented in Fig. 5.

As can be observed, the best results are achieved using the Random Forest algorithm, independently of segment and overlap sizes. Another observation is that, independently of the algorithm, for the same segment size, the resulting classifier quality increases with increasing overlap size. In addition, the quality of the classifier also increases with the increasing of the segment size.

A single F-score value related to a classifier (which represents the mean value for all classes of the classifier) is useful for comparing different classifiers. However, it does not show all the information about a classifier. To better understand the detailed behavior of a classifier it is necessary to evaluate the results for each class individually. Fig. 6 shows the classification level results for the 30/15, 60/30, 90/45 and 120/60 segment sizes (being the first number the segment size and the second number the overlap size). These are the four best classifiers found for each segment size. In the figure, for each class, the support (Sup) represents the number of occurrences of each class.

After analyzing the results of each class, we can say that the transport mode that is most easily identified is walking, with a minimum accuracy and recall of 0.98 and 0.96, respectively, for the segment size of 30 s. For bicycle and train, the results are also good, mainly for the segment size of 90 and 120 s. The worst identified transportation modes are car and bus. For the segment size with 120 s, the car has the precision of 0.89 and the recall 0.83, while the bus has the precision of 0.86 and the recall of 0.92.

Segment Size (s)	Overlap Size (s)	Decision Tree F-score (%)	Random Forest F-score (%)
30	0	76.87	82.84
	10	77.76	83.79
	15	78.54	84.64
60	0	79.02	85.68
	20	80.49	86.65
	30	80.76	87.45
90	0	80.85	87.57
	30	82.42	88.53
	45	82.95	89.61
120	0	82.67	88.48
	40	83.19	89.87
	60	84.36	90.63

Fig. 5. Results obtained for different sizes of segment and overlap, and for the two different algorithms being considered.

30/15	P	R	F	Sup
walking	0.98	0.96	0.97	5858
bicycle	0.86	0.88	0.87	2652
car	0.83	0.76	0.80	9420
bus	0.78	0.86	0.82	9014
train	0.84	0.80	0.82	2961
average	0.85	0.85	0.85	29905

60/30	P	R	F	Sup
walking	0.99	0.97	0.98	3022
bicycle	0.89	0.94	0.92	1428
car	0.86	0.80	0.83	5019
bus	0.82	0.89	0.85	4889
train	0.88	0.84	0.86	1656
average	0.88	0.87	0.87	16014

90/45	P	R	F	Sup
walking	0.99	0.97	0.98	1912
bicycle	0.92	0.95	0.94	951
car	0.89	0.82	0.85	3309
bus	0.84	0.91	0.88	3286
train	0.91	0.89	0.90	1111
average	0.90	0.90	0.90	10569

120/60	P	R	F	Sup
walking	0.99	0.98	0.98	1371
bicycle	0.93	0.97	0.95	702
car	0.89	0.83	0.86	2457
bus	0.86	0.92	0.89	2457
train	0.94	0.91	0.92	854
average	0.91	0.91	0.91	7841

Fig. 6. Random Forest results per class for the segment sizes: 30/15, 60/30, 90/45, and 120/60, with precision (P), recall (R), F-score (F), and support (Sup).

	Identified by the Classifier				
True Mode	walking	bicycle	car	bus	train
walking	97.28	1.52	0.63	0.36	0.21
bicycle	2.10	95.48	2.00	0.42	0
car	0.12	1.30	82.11	14.63	1.84
bus	0	0.15	7.55	91.45	0.85
train	0	0	5.49	5.85	88.66

Fig. 7. Confusion matrix (in %) for the Random Forest classifier with 90/45 (segment/overlap sizes).

Classifier	Trips with Errors	Non Existent Modes	Not Identified Modes
C1	12 (9.23%)	10	7
C2	7 (5.38%)	7	15
C3	21 (16.15%)	8	16

Fig. 8. Results of correct identification of the number of transport modes used by TMDM.

To better understand the lower results regarding the transport modes car and bus, Fig. 7 shows the confusion matrix for segment and overlap sizes of 90/45. Approximately 14.63% of the segments with true transport mode car were identified by the classifier as being bus. For segments with true transport mode bus, the classifier identified approximately 7.55% of the segments as corresponding to the transport mode car. Thus, taking these values into account, we can say that the classifier has some difficulties to distinguish the transport modes car and bus (being the most erroneous case the one in which the true transport mode is car). In fact, both transport modes have similar speeds and accelerations (motorized vehicles) making it hard to differentiate them. We can see that other transport modes are correctly identified.

5.1.2. TMDM

The TMDM was evaluated with trips from the test set composed by 130 trips with 371 transport modes used. In addition, to identify correctly the transport modes used, first the TMDM needs to identify the number of transport modes used. We selected the three best classifiers, all based on Random Forest, and named them C1, C2 and C3, parameterized with different segment and overlap sizes (C1 — 60/30 s, C2 — 90/45 s, C3 — 120/60 s).

The results related to the correct identification of the number of transport modes used are presented in Fig. 8. For each classifier used, the figure shows the number of trips with incorrect number of identified trips, the number of errors related with identification of nonexistent modes of transport, and the number of not identified modes of transport.

C3, which has the best results during the classifier evaluation, obtained the worst results when it was used in TMDM. About 16.15% of the trips had at least one error related with the number of transport modes used. The main cause of this error rate is the 120 s segment size, which is too large, and fails to identify the transport modes used for short periods of time.

Using C1, the number of misses with error is lower (9.23%), and there are more errors related to non existent modes. This can be explained by the lower quality of the classifier C1, i.e., there are more segments whose transport mode was wrongly identified; therefore, the TMDM identifies more transport modes that did not exist during the trip.

The best results were obtained using C2 which identified well the number of modes of transport used in 123 trips (94.62%) and missed only 7 trips (5.38%). The results obtained for still mode, using the classifier C2 were 0.96 for precision, 1.0 for recall and 0.98 for F-score with support of 24. C2 was considered to be the best being selected to be used in TMDM.

5.2. Final edgeTrans version

This section presents the evaluation results of the edgeTrans app when being used in the normal case, i.e., after the training phase has finished and the classifier generated. As already mentioned, the edgeTrans app (called Woorti) can be freely downloaded from the Android Play Store and the iOS iTunes, and it is available in 12 different languages (www.woorti.com). The app is currently being used by a wide set of users, mainly partners of the EU MoTiV project (motivproject.eu), but it can be used by any other user. In this section we present the evaluation of the battery consumption on Android and iOS, the accuracy obtained by the edgeTrans app when identifying transport modes, and the amount of data (opportunistically) sent from a smartphone. With respect to edgeTrans responsiveness, the information regarding the transport used by a smartphone's owner takes a maximum of 30 min, as this is the criteria to automatically detect that a trip has finished (as mentioned in Section 3.2.1).

5.2.1. Battery consumption

To understand the impact that the transport mode identification has on the battery life of a smartphone, we performed a set of tests to estimate the battery spent by the TMDM. The evaluation was done using the classifier C2 in the TMDM, which is the best classifier found; it uses a segment size of 90 s, and an overlap size of 45 s. To perform the battery tests,

Transport Mode	Num. of legs	Success Rate A	Success Rate B	Success Rate A (dist)	Success Rate B (dist)
walking	76565	90.14%	77.79%	85.79%	77.53%
bicycle	24529	80.48%	47.39%	95.54%	60.87%
car	36918	53.86%	77.30%	77.19%	80.19%
bus	9783	21.16%	33.92%	53.71%	49.61%
train	10241	50.47%	43.73%	91.68%	90.77%

Fig. 9. edgeTrans real-world usage results.

we used three different smartphones: both S1 (Xiaomi Redmi 5) and S2 (Xiaomi Redmi 2) run Android with 3300 mAh and 2200 mAh of battery capacity each, respectively; and S3 (iPhone 5S) running iOS with 1560 mAh of battery capacity.

During the tests, only the TMDM component of the edgeTrans app was running, and both networking connectivity and the screen were turned off. It is important to note that, during a trip, battery energy is always used by other processes; thus, the TMDM in the edgeTrans app uses less energy than the total energy that is effectively consumed. In each test, the battery level was observed using the information provided by the operating system.

In the first test, S1 and S2 were used in a trip that lasted for 1 h. On each smartphone, the battery level was measured at the beginning and at the end of the trip, to find the corresponding difference. On S1 the difference was 2% whereas on S2 the difference was 3%; this results in an average of 2.5% of battery use per hour. In the second test, it was used S1 in a 2 h trip. The battery difference was 4%, which results in an average of 2% of battery use per hour. In the third test, it was used S3 in a 2 h trip, as well. After 2 h of trip, the difference of battery level was 20%, i.e., 10% of battery use per hour.

For the Android operating system, the first test shows that, for the smartphones being considered, the percentage of battery spent per hour was similar with an average of 2.5%. The second test shows that for a larger trip, the percentage of battery spent per hour was also similar and equal to 2%. For the iOS, the results obtained were worse with 10% of battery spent per hour. A possible explanation for this difference is the battery health of the smartphone itself. In fact, such batteries lose their capacity over time by performing successive charges and discharges. The smartphone S3 is not as new as the other two; it is an old smartphone and suffered many cycles of charge and discharge, and even when performing basic tasks, the battery is consumed very fast. Energy consumption should be lower in other iOS devices with good battery health.

5.2.2. Detection of transport modes

Overall, we currently have 3422 active users (with at least one trip submitted) and 71058 submitted trips. The results concern the full operational usage of the edgeTrans app since November 2018, limited to partner research labs and, from April 2019, for the open usage and field survey mainly on cities across ten countries (Belgium, Croatia, Finland, France, Italy, Norway, Portugal, Slovakia, Spain, Switzerland). For all these users and trips we have many statistics available. For lack of space, we only show (see Fig. 9) the most relevant ones:[6]

Sucess Rate A = number of legs confirmed as X when the detected mode was X/Total number of legs detected as X;
Sucess Rate B = number of legs detected as X when the confirmed mode was X/Total number of legs confirmed as X;
Sucess Rate A (dist) = distance confirmed as X when the detected mode was X/Distance detected as X; and
Sucess Rate B (dist) = distance detected as X when the confirmed mode was X/Distance confirmed as X.

Note that these results were obtained in a real-world setting. Thus, contrary to most comparable works (mentioned in Section 2) which are either small-sized or use a simulated environment, we present results from real users using real transports with a real app running in real smartphones. This means that, globally, the accuracy of some results are not as good as we would like for some specific transport modes. That is the case of the transport mode bus. The success rates A and B for the transport mode bus is rather low which means that edgeTrans is probably confusing it with some other transport mode. This is due to the fact that the speed and acceleration of buses and cars (in a urban setting) are very similar, and in certain cases, also similar to trains. The success rate B indicates that the car transport mode is well recognized; the same applies when detecting the transport mode walking.

[6] Detected mode is the transport mode that is automatically detected by the edgeTrans app. Confirmed mode is the transport mode indicated manually by the user. A leg is simply a sequence of consecutive segments with the same transport mode.

Regarding the bicycle transport mode, the success rate A is not as high as for walking (even though it is above 80%) and the success rate B is not very high; this is due to the fact that in some places, electric bicycles are being used (in addition to classical non-electric bicycles), and the edgeTrans system was mainly trained with classical bicycles. When comparing electric with non-electric bicycles, clearly the speed and acceleration of the later are smaller.

When considering the transport mode train, we can say that sometimes this is confused with the car and bus transport modes. However, we can see the values for both success rates A (dist) and B (dist) are very good when the transport mode is train. This results from the fact that users, when traveling by train, make long trips for long distances.

Thus, globally, the values for both success rates A (dist) and B (dist), are much better that success rates A and B. This suggests that edgeTrans behaves better at identifying longer, and more relevant and meaningful, legs. In other words, edgeTrans seems to be identifying too many small legs which in fact do not exist. Thus, as future work, we intend to improve the detection and identification of small legs. We believe this could improve significantly the values for both success rates A and B.

5.2.3. Data exchanged

This section shows the amount of data that is exchanged between the edgeTrans app running in a smartphone and the server. As already mentioned, note that the sending of this data does not have to be done for the transport mode to be detected (in the smartphone). This information is sent from the smartphone in a opportunistic way.

The amount of the data depends on the time a user travels. For the edgeTrans app running in Android, the amount of data is 128 bytes/second, and for iOS the amount is 41.6 bytes/second. Thus, for example, if a user travels for 1 h in total, during a single day, and connects to the network at the end of the day (possibly at home), the total amount of data sent from the smartphone to the server is 460.800 bytes in Android, and 149.760 bytes in iOS.

6. Conclusions

This paper presents the edgeTrans system with an emphasis on the app that runs in a user's smartphone. This is a case in which the smartphone, i.e. an edge device, is used to run a classifier able to detect the transport mode being used, without requiring network connection and, therefore, in near real time and without requiring any server/cloud computing power.

We can say that the edgeTrans requirements are all respected. It is true that, in this paper, we focus on the requirements high accuracy, low battery consumption, and responsiveness; these are the only ones that we show quantitative results for. The other requirements are out of scope of this article; however, we believe that any reader can positively evaluate them by installing and running the edgeTrans app.

edgeTrans app is able to identify the transport mode used by a user carrying her/his smartphone with a classifier that thus runs at the edge. The edgeTrans app does not need to connect to the network to perform its task, therefore operating in a privacy-preserving manner with a low battery consumption. Finally, edgeTrans is being used by many users, in several cities around Europe, and provides a real-world setting which evaluation results are shown.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020 and by the MoTiV project, funded from the European Union's Horizon 2020 research and innovation programme under grant agreement № 770145.

References

[1] M. Nikolic, M. Bierlaire, Review of transportation mode detection approaches based on smartphone data, in: Swiss Transport Research Conference, Ascona, Switzerland, 2017, http://infoscience.epfl.ch/record/229181.

[2] M. Shoaib, S. Bosch, O.D. Incel, H. Scholten, P.J. Havinga, A survey of online activity recognition using mobile phones, Sensors 15 (1) (2015) 2059–2085, http://www.mdpi.com/1424-8220/15/1/2059.

[3] O.C. Ann, L.B. Theng, Human activity recognition: A review, in: 2014 IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2014, 2014, pp. 389–393, https://ieeexplore.ieee.org/document/7072750.

[4] M.A. Ayu, S.A. Ismail, A.F.A. Matin, T. Mantoro, A comparison study of classifier algorithms for mobile-phone's accelerometer based activity recognition, Procedia Eng. 41 (2012) 224–229, International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012). http://www.sciencedirect.com/science/article/pii/S1877705812025520.

[5] Global Mobile Consumer Trends (By Deloitte), Tech. rep., second ed., 2017, https://www2.deloitte.com/il/en/pages/technology-media-and-telecom\discretionary{-}{}{}munications/articles/global-mobile-consumer-survey.html.

[6] J. van Dijk, Identifying activity-travel points from GPS-data with multiple moving windows, Comput. Environ. Urban Syst. 70 (2018) 84–101, http://www.sciencedirect.com/science/article/pii/S0198971517304465.

[7] C. Zhou, H. Jia, J. Gao, L. Yang, F. Yixiong, G. Tian, Travel mode detection method based on big smartphone global positioning system tracking data, Adv. Mech. Eng. 9 (2017) http://dx.doi.org/10.1177/1687814017708134.
[8] A.C. Prelipcean, G. Gidófalvi, Y.O. Susilo, Transportation mode detection–an in-depth review of applicability and reliability, Transp. Rev. 37 (4) (2017) 442–464, https://www.researchgate.net/publication/309453453_Transportation_mo-de_detection_-_an_in-depth_review_of_applicability_and_reliability.
[9] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, M. Srivastava, Using mobile phones to determine transportation modes, ACM Trans. Sens. Netw. 6 (2) (2010) 13:1–13:27, http://dx.doi.org/10.1145/1689239.1689243.
[10] J. Zhou, Z. Cao, X. Dong, A.V. Vasilakos, Security and privacy for cloud-based IoT: Challenges, IEEE Commun. Mag. 55 (1) (2017) 26–33.
[11] M. Allamanis, E.T. Barr, P. Devanbu, C. Sutton, A survey of machine learning for big code and naturalness, ACM Comput. Surv. 51 (4) (2018) 81:1–81:37, http://dx.doi.org/10.1145/3212695.
[12] H. Schildt, Java: The Complete Reference, eleventh ed., McGraw-Hill Education Group, 2014, https://www.amazon.com/Java-Complete-Reference-Herbert-Schildt/dp/1260440230.
[13] M. Neuburg, IOS 8 Programming Fundamentals with Swift: Xcode and Cocoa Basics, first ed., O'Reilly Media, Inc., 2015, https://www.amazon.com/iOS-Programming-Fundamentals-Swift-Basics/dp/1491908904.
[14] D. Crockford, JavaScript: The Good Parts, O'Reilly Media, Inc., 2008, http://shop.oreilly.com/product/9780596517748.do.
[15] L.M. Surhone, M.T. Tennoe, S.F. Henssonow, Node.Js, Betascript Publishing, Beau Bassin, MUS, 2010, https://dl.acm.org/doi/book/10.5555/1941165.
[16] D. Maharry, TypeScript Revealed, first ed., Apress, Berkely, CA, USA, 2013, https://www.amazon.com/TypeScript-Revealed-Dan-Maharry/dp/1430257253.
[17] R. Sahay, Getting Started with Angular JS, CreateSpace Independent Publishing Platform, USA, 2015, https://www.amazon.com/Getting-Started-Angular-Rahul-Sahay/dp/1515037681.
[18] A performance evaluation of NoSQL databases to manage proteomics data, Int. J. Data Min. Bioinform. 21 (1) 70–89, https://dl.acm.org/doi/10.5555/3292787.3292792.
[19] A. Messina, P. Storniolo, A. Urso, EXPO: A modern NoSQL-based ebXML registry for health information systems, in: Proceedings of the 17th International Conference on Computer Systems and Technologies 2016, CompSysTech '16, ACM, New York, NY, USA, 2016, pp. 253–260, https://dl.acm.org/doi/pdf/10.1145/2983468.2983498.
[20] A. Bulling, U. Blanke, B. Schiele, A tutorial on human activity recognition using body-worn inertial sensors, ACM Comput. Surv. 46 (3) (2014) 33:1–33:33, http://dx.doi.org/10.1145/2499621.
[21] P.R. Stopher, S.P. Greaves, Household travel surveys: Where are we going? Transp. Res. A 41 (5) (2007) 367–381, http://dx.doi.org/10.1016/j.tra.2006.09.005, Bridging Research and Practice: A Synthesis of Best Practices in Travel Demand Modeling.
[22] S. Bricka, C.R. Bhat, Comparative analysis of global positioning system-based and travel survey-based data, Transp. Res. Rec. 1972 (1) (2006) 9–20, https://journals.sagepub.com/doi/10.1177/0361198106197200102.
[23] J. Wolf, Applications of new technologies in travel surveys, in: Travel Survey Methods, pp. 531–544, Ch. 29, https://www.emeraldinsight.com/doi/abs/10.1108/9780080464015-029.
[24] P. Stopher, C. FitzGerald, M. Xu, Assessing the accuracy of the Sydney household travel survey with GPS, Transportation 34 (6) (2007) 723–741, http://dx.doi.org/10.1007/s11116-007-9126-8.
[25] K. Deutsch, G. McKenzie, K. Janowicz, W. Li, Y. Hu, K.G. Goulias, Examining the use of smartphones for travel behavior data collection, in: 13th International Conference on Travel Behavior Research, Toronto, Canada, https://grantmckenzie.com/academics/McKenzie_IATBR2012.pdf.
[26] C.D. Cottrill, F.C. Pereira, F. Zhao, I.F. Dias, H.B. Lim, M.E. Ben-Akiva, P.C. Zegras, Future mobility survey: Experience in developing a smartphone-based travel survey in Singapore, Transp. Res. Rec. 2354 (1) (2013) 59–67, http://dx.doi.org/10.3141/2354-07.
[27] A.C. Prelipcean, G. Gidofalvi, Y.O. Susilo, Mobility collector, J. Locat. Based Serv. 8 (4) (2014) 229–255, http://dx.doi.org/10.1080/17489725.2014.973917.
[28] M.A. Shafique, E. Hato, Classification of travel data with multiple sensor information using random forest, Transp. Res. Procedia 22 (2017) 144–153, http://dx.doi.org/10.1016/j.trpro.2017.03.021.
[29] H. Xia, Y. Qiao, J. Jian, Y. Chang, Using smart phone sensors to detect transportation modes, Sensors 14 (11) (2014) 20843–20865, http://www.mdpi.com/1424-8220/14/11/20843.
[30] Y. Zheng, Q. Li, Y. Chen, X. Xie, W.-Y. Ma, Understanding mobility based on GPS data, in: Proceedings of the 10th International Conference on Ubiquitous Computing, UbiComp '08, ACM, New York, NY, USA, 2008, pp. 312–321, http://dx.doi.org/10.1145/1409635.1409677.
[31] C. Zhou, H. Jia, J. Gao, L. Yang, Y. Feng, G. Tian, Travel mode detection method based on big smartphone global positioning system tracking data, Adv. Mech. Eng. 9–6 (2017) http://dx.doi.org/10.1177/1687814017708134.
[32] B.D. Martin, V. Addona, J. Wolfson, G. Adomavicius, Y. Fan, Methods for real-time prediction of the mode of travel using smartphone-based GPS and accelerometer data, Sensors 17-9 (2017) http://www.mdpi.com/1424-8220/17/9/2058.
[33] A. Sauerländer-Biebl, E. Brockfeld, D. Suske, E. Melde, Evaluation of a transport mode detection using fuzzy rules, Transp. Res. Procedia 25 (2017) 591–602, http://dx.doi.org/10.1016/j.trpro.2017.05.444.
[34] L. Bao, S.S. Intille, Activity recognition from user-annotated acceleration data, in: A. Ferscha, F. Mattern (Eds.), Pervasive Computing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 1–17, https://link.springer.com/chapter/10.1007/978-3-540-24646-6_1.
[35] N. Krishnan, D. Colbry, C. Juillard, S. Panchanathan, Real time human activity recognition using tri-axial accelerometers, in: Proceedings of the Sensors, Signals and Information Processing Workshop, Sedona, AZ, USA, 2008, https://www.researchgate.net/publication/239920951.
[36] J.R. Kwapisz, G.M. Weiss, S.A. Moore, Activity recognition using cell phone accelerometers, SIGKDD Explor. Newsl. 12 (2) (2011) 74–82, http://dx.doi.org/10.1145/1964897.1964918.
[37] T. Brezmes, J.-L. Gorricho, J. Cotrina, Activity recognition from accelerometer data on a mobile phone, in: S. Omatu, M.P. Rocha, J. Bravo, F. Fernández, E. Corchado, A. Bustillo, J.M. Corchado (Eds.), Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 796–799, https://link.springer.com/chapter/10.1007/978-3-642-02481-8_120.
[38] N. Győrbíró, Á. Fábián, G. Hományi, An activity recognition system for mobile phones, Mob. Netw. Appl. 14 (1) (2009) 82–91, http://dx.doi.org/10.1007/s11036-008-0112-y.
[39] D.J. Patterson, L. Liao, D. Fox, H. Kautz, Inferring high-level behavior from low-level sensors, in: A.K. Dey, A. Schmidt, J.F. McCarthy (Eds.), UbiComp 2003: Ubiquitous Computing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 73–89, https://link.springer.com/chapter/10.1007/978-3-540-39653-6_6.
[40] T. Sohn, A. Varshavsky, A. LaMarca, M.Y. Chen, T. Choudhury, I. Smith, S. Consolvo, J. Hightower, W.G. Griswold, E. de Lara, Mobility detection using everyday GSM traces, in: P. Dourish, A. Friday (Eds.), UbiComp 2006: Ubiquitous Computing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 212–224, https://link.springer.com/chapter/10.1007/11853565_13.
[41] D.-N. Lu, D.-N. Nguyen, T.-H. Nguyen, H.-N. Nguyen, Vehicle mode and driving activity detection based on analyzing sensor data of smartphones, Sensors 18–4 (2018) http://www.mdpi.com/1424-8220/18/4/1036.
[42] A. Guazzelli, M. Zeller, W.-C. Lin, G. Williams, PMML: An open standard for sharing models, R J. 1 (1) (2009) 60–65, http://dx.doi.org/10.32614/RJ-2009-010.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830, http://dl.acm.org/citation.cfm?id=1953048.2078195.

[44] M. Sokolova, G. Lapalme, A systematic analysis of performance measures for classification tasks, Inf. Process. Manage. 45 (4) (2009) 427–437, http://dx.doi.org/10.1016/j.ipm.2009.03.002.

[45] I. Guyon, J. Weston, S. Barnhill, V. Vapnik, Gene selection for cancer classification using support vector machines, Mach. Learn. 46 (1) (2002) 389–422, http://dx.doi.org/10.1023/A:1012487302797.