An Information System for Bus Travelling and Performance Evaluation

Leandro Ricardo[2], Susana Sargento[1,2] and Ilídio C. Oliveira[1,3]
[1]*Departamento de Eletrónica, Telecomunicações e Informática, Universidade de Aveiro, Campus Universitário de Santiago, Aveiro, Portugal*
[2]*Instituto de Telecomunicações, Campus Universitário de Santiago, Aveiro, Portugal*
[3]*Instituto de Engenharia Electrónica e Informática de Aveiro, Campus Universitário de Santiago, Aveiro, Portugal*

Keywords: Vehicular Networks, Long-Term Time Estimation, Prediction, Machine Learning.

Abstract: A wide vehicular network has a huge potential to collect city-data, specially with respect to city mobility, one of the top concerns of the municipalities. In this work, we propose the use of the mobility data generated by the movement of the connected buses to deliver a new set of tools to support both the bus passengers and bus fleet operator use cases. Considering the bus passengers, it is possible to build smart schedules, which deliver an estimated time of arrival based on the city dynamics along time, and that can be accessed directly in the smartphone. Considering the bus fleet operator, it is possible to characterize the behaviour of buses and bus lines. Using the GPS trace of buses and map-matching algorithm, we are able to discover the line each bus is assigned to. Estimated times of arrival and predictions are implemented recurring to time estimations and predictions, using both data mining and machine learning approaches. Proof-of-concept applications were implemented to demonstrate the real-life applicability, including a mobile app for the citizens, and a web dashboard for the fleet operator.

1 INTRODUCTION

In last several years, much has been told about *smart cities*. A smart city is a city which tries to bring the best out from the cooperation between people and policies through the use of Information and Communication Technologies, that connects and empowers both (Monzon, 2015).

Much like human communities, Smart cities can be sensed, analyzed and served. Sensing is possible using a wide sensor network, that can be either fixed, when it is located, for instance, in buildings or dynamic, as happens with sensors of vehicular networks. These sensors are generating a big amount of data everyday and that same data can be used for analyzing the city dynamics. The internet of the *"moving things"* is happening right now in Porto, a *Smart City* which deploys a world-level pilot project implementing a Vehicular Ad-hoc Network connecting buses, garbage trucks and taxis, and delivering free internet access to bus passengers, enhancing their commuting experience.

Considering the *Smart Transportation* topic, improving the mobility is possible by studying those bus lines flows in terms of their completion and delay. Also, considering passengers, com muting experience can be enhanced by providing time estimations (or predictions) concerning the arrival times in such a way that passengers can use this information anywhere, for instance, in their *smartphones*, for moving with agility through the city.

Regarding this, our study proposes a generic solution to build **arrival time estimations** and **predictions** from the data resulting from the movement of the wondering nodes of the vehicular network. For being able to perform arrival time estimations and predictions, a map-matching algorithm has been proposed for identifying which bus line is being completed by a given bus.

Three ensemble machine learning methods have been tested and compared. Finally, a set of services and proof-of-concept applications were done with the goal of showing the practical utility of the system for bus passengers and bus fleet managers as a decision support system.

This article starts by presenting studies which serve as basis of our work. Then, we present the proposed architecture for solving our problem and each one of its core components. The results of our study are discussed for a matter of self-evaluation of this work and finally, some conclusions are withdrawn.

Ricardo, L., Sargento, S. and C. Oliveira, I.
An Information System for Bus Travelling and Performance Evaluation.
DOI: 10.5220/0006707903950402
In *Proceedings of the 4th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2018)*, pages 395-402
ISBN: 978-989-758-293-6

2 RELATED WORK

The study driven by (As and Mine, 2016) starts defends that the most valuable resources which can be given to the bus passengers is the estimated time of arrival, an argument that is supported by the observation that, if bus passengers know *"their departure time and arrival time at the destination"* they can *"reduce their waiting time at the bus stop"*.

Under the same topic, (Uno et al., 2009) argument that the fast progress of the information technology are leading to new insights about traffic phenomena that can be solved. Also, they refer that the identification of particles moving through the city is one of the key areas of application on the traffic and transportation study areas. This work focuses on presenting a methodology for using GPS data with the aim of turning it meaningful to transportation analysis. It summarizes also methodologies and transformation techniques that can be applied to the GPS data. The final aim of this study is to propose an approach for evaluating the bus lines quality of service, from the point of view of the travel time stability and reliability.

Regarding time estimation and prediction, the review written by (Mori et al., 2015) presents the state of the art of this topic, going even further by explaining the main definitions and how the advanced traveller information systems work.

Finally, in terms of methods for long term travel time prediction, the study conducted by (Mendes-Moreira et al., 2012) highlights the importance of this prediction and how it can be an important measure for public transportation companies. They also make a comparison of three non-parametric popular regression methods, namely, Support Vector Machine, Random Forests and Projection Pursuit Regression using the data from the same public transportation company of our study. As remarking conclusions, Random Forest is elected as the best method and it is advised to use ensemble learning methods for improving the accuracy of the predictions.

This work borrows some important ideas from the study in (Uno et al., 2009), such as how to separate the pipeline (like map-matching, data reduction, data processing and data reporting) and the important functionalities on each of the stages.

The related work also shows that some studies are very different in nature, like the study conducted (As and Mine, 2016), where the granularity of the available data is larger (1 second) and the information of the route, number of bus stops, travel direction and bus performance history is present as probe data used for the study.

3 PROPOSED SYSTEM AND ARCHITECTURE

The proposed system architecture, represented in the figure 1, includes several modules and data sources to support the processing pipeline.

From the bottom of the diagram to the top, please find the description of each element:

- **vanetV3 and the STCP Website** are the main raw data sources. *vanetV*3 contains the location information of each bus every 15 seconds, gathered through the vehicular network, and the STCP website contains the information on bus lines and schedule.
- The **Extraction Scripts** transform the raw data, existing on the website, in a set of well-known and well structured files containing all the context information such as the bus lines, bus stops, etc.
- The **Matching Unit** consists in a Python program which conforms the log position data with the bus network data.
- The **Matches Database** is responsible for holding the Matching Unit results (bus line identification, matched bus stops, etc).
- The **Estimation Database** helps to gather performance metrics from bus delays.
- The **Synchronization Script** performs the synchronization between the Matches Database and the Estimation Database.
- The **Bus Network Information Database** holds information about the bus carrier such as line, bus stops and their relationship.
- **Bus Network Information API** delivers information about the bus carrier infrastructure (the lines, stops, the relation between them).
- **Matches API** delivers information about the matches, data which is stored in the *Matches Database*.
- **Estimation API** delivers only one endpoint for gathering the estimated times of arrival given a stop, a line and a date, data which is stored in the *estimation database*.
- **Prediction API** delivers an endpoint for making predictions, using the prediction module.
- The **Line Performance Dashboard** is a dashboard for the bus carrier manager to consult the performance of the carrier's lines. It is a decision support dashboard because it provides hints about the health of the bus transportation system.

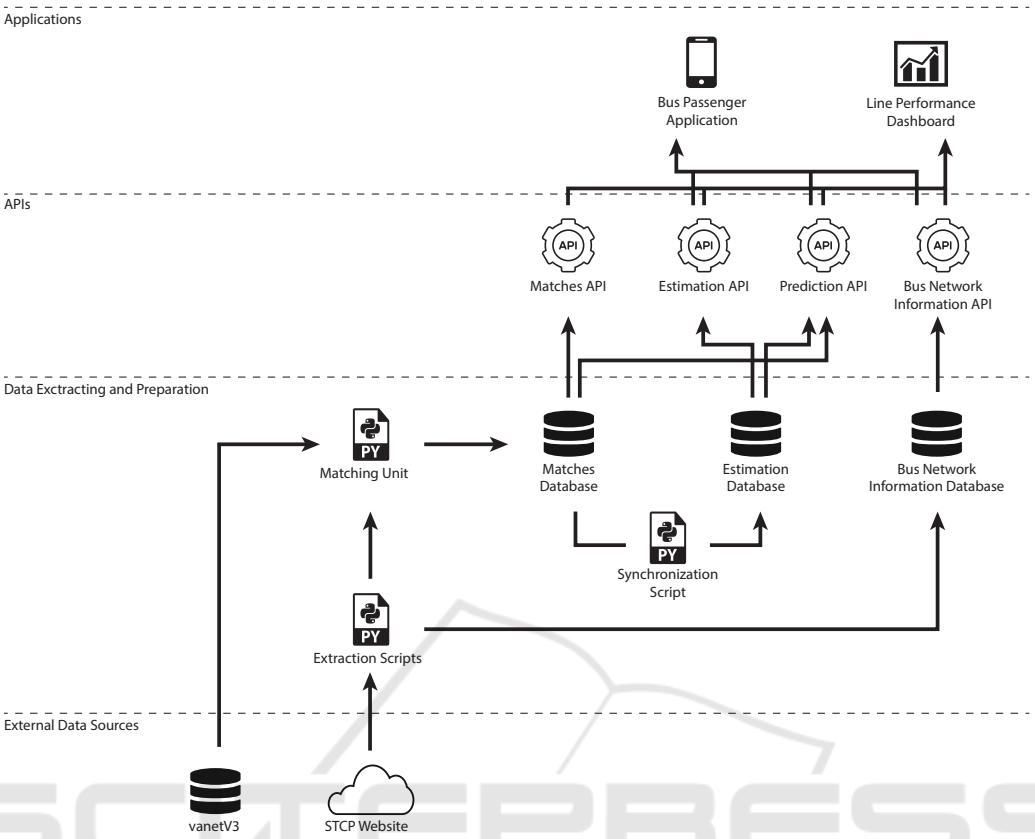

Figure 1: Architecture components and the data flow.

- The **Bus Passenger Application** is a mobile application meant to provide the estimated times of arrival for a given line and bus stop to the bus passengers (the carrier's end-users).

The following subsections explain in further detail the core components this solution.

3.1 Matching Algorithm Design

3.1.1 Overview

With the purpose of matching a set of GPS locations of a given bus with the bus line (set of bus stops) that it is performing, a map-matching algorithm is proposed. Notice that the bus assigned to each bus line is independent from the previous line performed in the same and in different days, and the only information available is the mobility of the bus.

The algorithm comprises two steps:

- The bus line start detection;
- The greedy search for validation of the candidate as a solution.

These two steps are explained in further detail on the next two subsections.

3.1.2 Bus Line Start Detection Algorithm

On each iteration of the algorithm, it is applied a heuristic, which consists on detecting the bus line starting on that given position.

To do that, for each of the GPS positions, bus stops are detected using a fixed-size radius that is calculated using the Vincenty Formulae (see (T. Vincenty, 1975)), and then, for each of the bus stops it is verified if they are the first bus stop of any of the bus lines.

The algorithm 1 implements the mechanism for detecting bus lines starts.

3.1.3 Bus line Completion Greedy Search Algorithm

After finding a bus line starting, the second step is to analyze whether or not the candidate is truly a solution.

The figure 2 presents the steps for validating a candidate bus line as solution. First, it is presented a sam-

Input: QueryTool: an object for making spatial queries
Input: Records: a set of records from a given node, in a given day
Output: *start_profile*: a list of the line start detections for each one of the records

```
start_profile ← ∅
foreach record ∈ Records do
    record_position ←
      record.extractLatitudeLongitude()
    nearest_stops ←
      QueryTool.getNearestBusStops(record_position)
    lines_of_the_record ← ∅
    foreach bus_stop ∈ nearest_stops do
        line_of_bus_stop ←
          QueryTool.getLinesStartingWith(bus_stop)
        lines_of_the_record.append(lines_of_bus_stop)
    start_profile.append(lines_of_the_record)
```

Algorithm 1: Detect Line Starts Algorithm.

Figure 2: Steps for validating a candidate bus line as solution.

ple of GPS positions, represented as blue dots. Then, it is presented two bus lines starting at the current GPS position being iterated by the algorithm. Then, the two remaining steps are implemented by the greedy search algorithm for validation of the bus line. On the third subfigure, in order to test if the candidate bus line is a solution, it is loaded and finally, tested, stop by stop, using a detection radius (marked in green).

The algorithm 2 implements the mechanism presented in the fourth subfigure of the figure 2.

As input, it takes four elements: a QueryTool object for making spatial queries, the position log database records to be matched, the set of starting lines for each one of the records, and finally, the base

```
i ← 0;
solutions ← ∅
while i <Records.length() do
    if StartingLines.index(i) is not empty then
        foreach candidate ∈ StartingLines.index(i) do
            j = i;
            line_stops ←
              QueryTool.getStopsFromLine(candidate);
            first_stop ← line_stops.getFirstElement();
            line_stops.setElement(0, None);
            last_stop ← line_stops.getLastElement();
            matches ← ∅;
            indexes ← ∅;
            solution_found ← False;
            target_order ← 1;
            ttl ← BaseTTL;
            while ttl >= 0 do
                if j >= Records.length() then
                    return solutions
                ttl ← ttl − 1;
                current_position ←
                  Records.index(j).getLatitudeLongitude();
                nearest_bus_stops ←
                  QueryTool.getNearestBusStop(current_position);
                foreach nearest_bus_stop ∈ nearest_bus_stops do
                    if nearest_bus_stop ∈ line_stops and
                      nearest_bus_stop ∉ matches then
                        stop_order ←
                          line_stops.index(nearest_bus_stop);
                        if stop_order ∈
                          [target_order, target_order + 5] then
                            matches.append(nearest_bus_stop);
                            indexes.append(j);
                            target_order ← stop_order + 1;
                            ttl ← BaseTTL;
                completeness ←
                  (matches.size() + 1)/line_stops.size();
                if last_stop ∈ matches and completeness >= 0.8
                  then
                    matches.insert(0, first_stop);
                    matches.insert(0, i);
                    solution ← initializeMap();
                    solution.map('line', candidate);
                    solution.map('stops', matches);
                    solution.map('indexes', indexes);
                    solution.map('completeness', completeness);
                    solutions.append(solution);
                    i ← j;
                    solution_found ← True;
                j ← j + 1;
                if solution_found is True then
                    break;
    i ← i + 1
```

Algorithm 2: Greedy search for validation of the candidate as a solution.

TTL, later explained.

Input: QueryTool: an object for making spatial queries
Input: Records: a set of records from a given node, in a given day
Input: StartingLines: a start profile of the the records, output from the procedure implementing the heuristic `detect_line_starts`
Input: BaseTTL: the timeout for finding a next bus stop (the default is 15 minutes)
Output: *solutions*: a list of matches

Iterating over all the sucessive GPS positions of a bus, the algorithm verifies if there are bus lines starting at the current GPS position. The variable *i* tracks the overall progress.

The method's input provides *StartingLines*, a list of bus lines starting at the current GPS position. Thus, we must test if any of them is locally a solution by simulating the bus course (line 5).

Before starting the simulation, a context is initialized. For tracking the current progress of the simulation, *j* is set. Then, the stops of the current line are loaded and the remaining context is built: `matches` holds the detected bus stops for the candidate line, `indexes` locate the bus stop appearance in time and `solution_found` controls the simulation, interrupting it when a solution is found. Finally, `target_order` represents the order of the next bus stop that the algorithm will try to detect (line 6 to 14). The initial target order is one.

The algorithm comprises also a `time-to-live` (or simply TTL). Therefore, it is able to give up on the current simulation if it is unable to find the target bus stop in a certain amount of time (15 minutes, based on empirical evidences). When `ttl` reaches zero, another candidate is tested.

The simulation itself starts at the line 16. The current GPS position is extracted out of the record (database element), and the nearest bus stops are retrieved (lines 20-21). For each of the bus stops, it is verified if they belong to the line and if they are not already matched. If they are not matched, the bus stop order is extracted and it is made a verification for asserting that the bus stop is expected in a near future.

If the order of the detected bus stop fits this criteria, both `matches`, `indexes` and the `target_order` are updated. The variable `ttl` is set to its default (lines 24-29).

After all the bus stops finish the evaluation, the completeness rate is computed. If the last stop was detected and the completeness is larger than 80%, the solution is saved into a data structure and added to the solutions. The lines 32 and 33 handle the detection of the first stop, which is the first to be detected but the last to be tracked. This is also implied as technique for detecting circular lines since, if the completeness metric is targeted and the last stop is detected, only one bus stop will be missed (the first, regarding the initial target order). The iteration variable `i` is updated, making the main control cycle ignore the analysis of all the intermediate records which are covered by the found solution (lines 32-41).

3.2 Estimating Arrival Times

After the system being capable of identifying which bus line has a bus completed, it is possible to use the data from the source database to build statistics about the bus line arrival times. Unlike most of the literature which makes use of times between stops to calculate time estimations, our study uses passing times because they require less computation power and also because the data granularity is very low (15 seconds).

The process of estimating arrival times consists in the following steps:

1. Choosing a bus line and a bus stop.
2. Finding a reference time (for instance, a timestamp chosen by the user or a timestamp gathered from a fixed time table of the carrier).
3. Then, over all the found matches in a period of time (for instance, one month), find the matches that fit the previous captured timestamp in a given reference window (for instance, 5 minutes around the reference time).
4. Obtain those corresponding timestamps.
5. Compute the average or the median of the gathered timestamps.

3.3 Predicting Arrival Times with Machine Learning

The goal of the prediction module is to deliver predictions of bus arrival times at each bus stop of a bus line. Thus, this module can be integrated in a service that can be accessed by bus passengers, in real-time.

Using machine learning is a natural approach for implementing this module. This is a supervised learning task because training values are provided as samples (the timestamps). Since the target value is not a class but rather a continuous value (a float expressing a time in the form of UNIX epoch), the type of machine learning algorithms that need to be tested are regressions.

The three tested algorithms are the following:

- **Random Forrest Regressor**[1]
- **Gradient Boosting Regressor**[2]
- **Bagging Regressor (using Decision Trees or Support Vector Regression)**[3]

These machine learning algorithms are ensemble methods, as recommended by ((Mendes-Moreira et al., 2012)), in order to obtain a better accuracy.

The objective of using three different algorithms is to find which one of them performs better. The prediction module is implented using the `Python` language and the framework `scikit-learn`.

3.4 Overall Integration

3.4.1 APIs

The information generated by the matching algorithm is stored into a database. Also, the estimation mechanism is implemented in the database level.

Since the application does not interact directly with the databases, an abstraction level is provided through the provision of a set of REST APIs. They are implemented using the `Python` language and `connexion`[4], which provides a mean for presenting the API documentation automatically from a `swagger`[5] YAML[6] specification file.

3.4.2 Applications

Two main applications have been developed as an effort to illustrate the outputs of the system architecture. One is a bus line performance dashboard, meant for a bus carrier manager and the other is a mobile application for bus passengers.

Illustrations of these applications are shown in the figures 3 and 4.

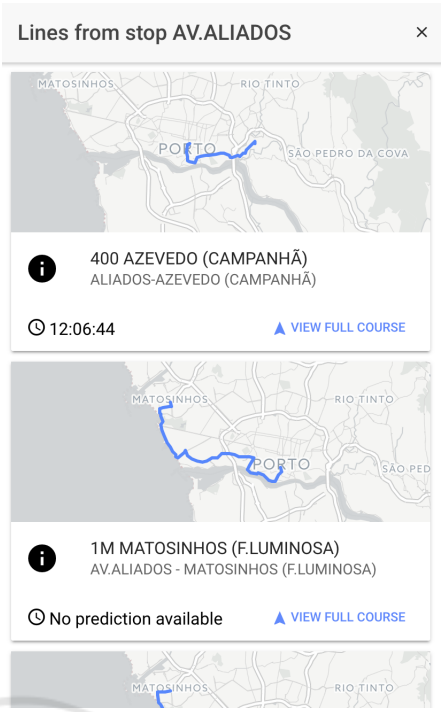

Figure 3: Estimated Times of Arrival for a given bus stop.

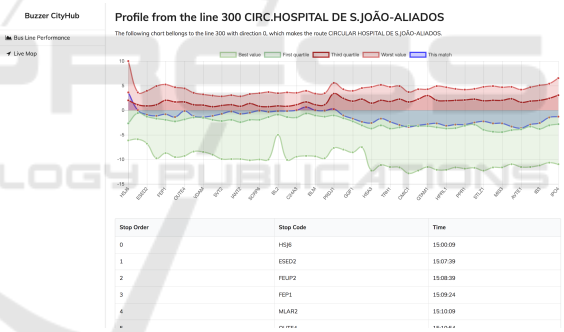

Figure 4: Decision Support Dashboard showing a delay plot.

4 RESULTS

4.1 Map-matching Results

A natural question, at this stage, is how many bus lines were matched in the period of this study (since the start of January until the end of March). Below, we present some numbers:

- **Number of SQL Records Processed:** 94 328 959.
- **Average Processing Time (Depends on the Dataset):** 35h14m per month.
- **Number of Detected Bus Lines:** 268 702, totaling 2985 bus lines completions per day.

[1]Random Forrest Regressor on `scikit-learn` documentation (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html)

[2]Gradient Boosting Regressor on `scikit-learn` documentation (http://scikit-learn.org/stable/modules/generated/ sklearn.ensemble.GradientBoostingRegressor.html#examples-using-sklearn-ensemble-gradientboostingregressor)

[3]Bagging Regressor on `scikit-learn` documentation (http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html)

[4]Connexion, a Swagger/OpenAPI First framework for Python on top of Flask with automatic endpoint validation & OAuth2 support (https://github.com/zalando/connexion)

[5]Swagger, The world's most popular API tooling (https://swagger.io/)

[6]YAML Ain't Markup Language (http://yaml.org/)

- **Number of Detected Bus Stops:** 9 460 737, totalling 105 119 bus stops detection per day.
- **Average of the Completion Rate:** 97.199(89) %.

The three months of captures result in more than 94 million SQL records with the location of the buses. With this technique, 268 thousand matches were found, adding value to `vanetV3`.

The average completion rate, despite of not being a totally reliable metric, shows that in the average, the overall bus lines detected have 97.2% of its bus stops detected, being a strong indicator that the algorithm works mostly well.

4.2 Time Estimation Results

The delay plot of a bus line is presented on the Line Performance Dashboard. The *delay plot* helps us on understanding how a bus line usually behaves during the bus course completion as well as on understanding if the bus line completion usually completes uniformly or if it varies locally (next to a bus stop or a set of bus stops). Also, the delay plot provides a mean for studying the behavioral differences in terms of delay of a given bus line on rush and non-rush hour periods while enabling long-term time estimations (for instance, in 3 months).

Five values are presented: the current match (in blue) represents the current bus line completion that is being compared, the inner red line represents the value of the third quartile (75%) and the inner green represents the first quartile (25%). These metrics are represented towards the median value (second quartile, with value 50%), defined as zero. The outer values sample the edges of the distribution. Values higher than zero are said to be late in relation to the distribution.

Regarding these characteristics, we have chosen some of the bus lines of STCP that seem to reveal traffic patterns or mobility problems. For instance, the bus line `204 - HOSPITAL DE SAO JOÃO` reveals very well the traffic and delay patterns. It comprises some popular points of interest, such as *Faculdade de Engenharia da Universidade do Porto*, *Escola Superior de Educação* and *Hospital de São João*.

Having such destinations, it is a line that is worth studying because of its impact on student , teachers and other faculty members mobility.

Below, we present a delay plot of the bus line `204 - HOSPITAL DE SAO JOÃO` at February 20, using the period from January 20th until February 20th of 2017 as reference. The delay plot refers to the period between `07:57` until `09:02 a.m`, during the rush hour. The *x-axis* represents the bus stop codes and the *y-axis* represents the time in minutes.

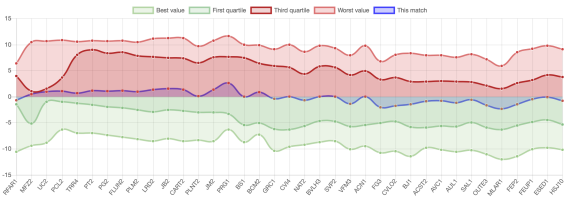

Figure 5: Line 204 Hospital de S. João from `07:57:50` until `09:02:20`.

4.3 Machine Learning Results

For evaluating the model with newly found hyperparameters, regression metrics needed to be calculated. The machine learning framework's API provides five regression metrics: the Explained Variance Score (EVS), Mean Absolute Error (MAE), Mean Squared Error (MSE), Mean Squared Logarithmic Error (MSLE), Mean Squared logarithmic error and Median Absolute Error (Median AE), Determination Coefficient (R^2).

The table 1 presents these metrics comparison. More information about their meaning can be consulted on Scikit-Learn Documentation [7] about the regression metrics.

Table 1: Resulting Evaluation Metrics.

	Bagging (Decision Tree Estimator)	Bagging (Support Vector Regressor)	Gradient Boosting	Random Forrest
EVS	0.982	0.975	0.985	0.984
MAE	179.999	133.562	183.049	183.319
MSE	38972.588	22614.798	39265.194	39925.414
MSLE	1.309e-05	7.682e-6	1.320e-05	1.342e-05
MAE	201.664	130.909	206.808	206.723
R^2	0.899	0.941	0.898	0.897

Overall, using the bagging algorithm with SVR as base estimator is better choice because:

- The **determination coefficient** R^2 is higher, meaning that it is likely that this model is better on predicting accurately;
- The **mean square error, mean square logarithmic error, and the median absolute errors** are almost half lower than the counterpart models;
- The **model is more general**: when plotting the solution, the generated regression is mathematically simpler than the counterparts;

Therefore, the model chosen for being used in the Prediction API is Bagging with SVR as base estimator.

An important note is that the median errors (mean absolute errors) are approximately 130 seconds, meaning that predicted passing times may have

[7] Regression Metrics (http://scikit-learn.org/stable/modules/classes.html#regression-metrics)

an error in the order of 2 minutes and 10 seconds. This is subject to change, depending on the data variance, something that needs to be studied in more depth, in the future.

5 CONCLUSIONS

This paper proposed an approach to generate meaningful information for bus passengers, municipalities and transportation companies.

The match algorithm is a generic solution, depending on few data to deliver a matching solution. With it, it is possible to guess what bus line is a bus traversing, not only in Porto, but also in other places, if the vehicular network exist and also information regarding bus carrier network is provided.

Then, the arrival time estimation delivers foundations for using as recommendation for bus passenger or, for delivering reports and decision support systems for the bus fleet operators.

Also prediction is introduced, exploring ensemble learning as a measure for improving the accuracy of the literature studied methods. The estimators perform in a similar way but the best performing one is bagging, using support vector regressor as base estimator, due to being more general and providing a lower error.

Finally, the applications for bus passengers and bus fleet operators are a proof of the utility provided by this whole architecture. Future work will provide real assessment of these applications.

ACKNOWLEDGEMENTS

This work was supported by the CMU-Portugal Program through S2MovingCity: Sensing and Serving a Moving City under Grant CMUPERI/TIC/0010/2014.

REFERENCES

As, M. and Mine, T. (2016). Empirical Study of Travel Time Variability Using Bus Probe Data. In *2016 IEEE International Conference on Agents (ICA)*, pages 146–149. IEEE.

Mendes-Moreira, J., Jorge, A. M., De Sousa, J. F., and Soares, C. (2012). Comparing state-of-the-art regression methods for long term travel time prediction. *Intelligent Data Analysis*, 16(3):427–449.

Monzon, A. (2015). Smart Cities and Green ICT Systems (SMARTGREENS), 2015 International Conference on. *Smart Cities and Green ICT Systems (SMARTGREENS), 2015 International Conference on*, pages 1–11.

Mori, U., Mendiburu, A., Álvarez, M., and Lozano, J. A. (2015). A review of travel time estimation and forecasting for Advanced Traveller Information Systems. *Transportmetrica A: Transport Science*, 11(2):119–157.

T. Vincenty (1975). Direct and Inverse solutions of geodesics on the ellipsoid with application of nested equations. *Survey Review*, XXII(176).

Uno, N., Kurauchi, F., Tamura, H., and Iida, Y. (2009). Using Bus Probe Data for Analysis of Travel Time Variability. *Journal of Intelligent Transportation Systems*, 13(1):2–15.