Public Transport Arrival Time Prediction based on GTFS data

Eva Chondrodima[1][0000−0002−4433−0525], Harris Georgiou[1][0000−0003−3462−0745], Nikos Pelekis[2][0000−0001−7205−5703], and Yannis Theodoridis[1][0000−0003−2589−7881]

[1] Dept. of Informatics, University of Piraeus, Piraeus, Greece
{evachon, hgeorgiou, ytheod}@unipi.gr
[2] Dept. of Statistics and Ins. Sci., University of Piraeus, Piraeus, Greece
npelekis@unipi.gr

Abstract. Public transport (PT) systems are essential to human mobility. PT investments continue to grow, in order to improve PT services. Accurate PT arrival time prediction (PT-ATP) is vital for PT systems delivering an attractive service, since the waiting experience for urban residents is an urgent problem to be solved. However, accurate PT-ATP is a challenging task due to the fact that urban traffic conditions are complex and changeable. Nowadays thousands of PT agencies publish their public transportation route and timetable information with the General Transit Feed Specification (GTFS) as the standard open format. Such data provide new opportunities for using the data-driven approaches to provide effective bus information system. This paper proposes a new framework to address the PT-ATP problem by using GTFS data. Also, an overview of various ML models for PT-ATP purposes is presented, along with the insightful findings through the comparison procedure based on real GTFS datasets. The results showed that the neural network -based method outperforms its rivals in terms of prediction accuracy.

Keywords: estimated time of arrival · GTFS · GTFS-RT · GTFS validation · machine learning methods · mobility data mining · neural networks · public transport

1 Introduction

Public transport (PT) offers significant social and environmental benefits. More specifically, high quality PT services lead to: (a) a considerable improvement on the quality of citizens' life, e.g. areas with access to public transportation help social inclusion, and (b) environmental benefits related to minimizing the CO2 emissions of private vehicles. Public transportation goal is to provide efficient, reliable, and high quality services, in order to attract more passengers. The planning of high quality PT systems is a difficult task. PT networks are highly complex systems, due to the large number of passengers that are transported

each day, due to the number of employees and because they are affected by technical and organizational complications.

The wide adoption of GPS tracking systems in PT provides new opportunities for using data-driven approaches to fit the demand of passengers and provide effective bus information system. Arrival time prediction (ATP) of PT vehicles is an important part of intelligent transportation systems. Accurate prediction can assist passengers in planning their travels and may improve travel efficiency. Also, efficient PT-ATP is necessary in order to eliminate passengers' long waiting time for the arrival of a new vehicle and the existence of delays during a trip. However, accurate PT-ATP is a challenging task due to a variety of factors, including stochastic variables such as traffic conditions, weather conditions, etc.

In order to deal with the PT-ATP problem it is necessary to collect moving PT vehicles information, manage big mobility data, and address spatio-temporal prediction problems. The collection and data management related to PT vehicles is accomplished through PT agencies. However, the inherent difficulty of managing PT data poses challenges in terms of storing data in Big Data platforms, as well as further analysis to extract useful and usable knowledge. Indeed, valuable knowledge is hidden in big mobility data, which can be fully exploited through Machine Learning (ML) techniques, such as Neural Networks (NN).

The key to the implementation and validation of transport models are the real-world data. Their availability and quality can significantly affect the reliability of the resulting estimates [3]. Data availability and data quality are of equal importance. Nowadays thousands of PT providers employ a common format for publishing their public transportation schedules and associated geographic information, called General Transit Feed Specification (GTFS). GTFS data are composed of two types of feeds: a) the GTFS feed (also known as static GTFS), which contains static timetabling information, and b) the GTFS-real time (GTFS-RT) feed, which contains real time information about the transit network.

However, processing PT raw data, even standard GTFS, is challenging. Particularly, GTFS data (static and real-time) often contain missing information and errors, such as missing timetable information (e.g. times of operations), invalid stops coordinates, invalid vehicle coordinates, e.t.c. Thus, GTFS data should pass through a set of validation steps. In order to validate GTFS and GTFS-RT feeds, open-sources have been introduced [11]. However, GTFS validator tools cannot guarantee that the validated data are appropriate for using ML methods for ATP purposes. To address this problem and to make GTFS and GTFS-RT feeds appropriate for learning ATP purposes, we introduce a tool for Cleansing and Reconstructing GTFS data, called CR-GTFS tool.

In the previous years, a number of prediction algorithms have been applied to moving objects [6–8, 20]. Furthermore, various studies have been conducted that use ML techniques in predicting the transit travel time by using GPS traces from transport vehicles [9, 16, 28], or the so-called Live Automatic Vehicle Locations (AVL) data [12, 19, 21]. There are also some works that employ AVL with GTFS feed. In [15] the purpose was to train an NN to predict the travel times

of buses based on open data collected in real-time, while the model evaluation was conducted on data derived from Sao Paulo City bus fleet location, real-time traffic data, and traffic forecast from Google Maps. By mining Live AVL data that buses provided by the Toronto Transit Commission along with schedules retrieved from GTFS, and weather data, Alam et.al [1] found that their proposed recurrent NN-based architecture predicts the occurrence of arrival time irregularities accurately. In the literature there are limited works regarding the PT-ATP problem by using GTFS and GTFS-RT feeds. Particularly, Sun et al. [24] combined clustering analysis with Kalman filters to predict arrival times at various bus stops on Nashville, TN, USA, by calculating the delay versus a scheduled time, based on GTFS and GTFS-RT as well as historical bus timing data.

The main contribution of this work is to propose a new framework to clean-reconstruct GTFS and GTFS-RT feeds and simultaneously address the PT-ATP problem by using GTFS data. Also, we examine various ML models for PT-ATP purposes and provide insightful findings through the comparison procedure.

The rest of this paper is organized as follows: Section 2 presents the employed GTFS data along with the proposed GTFS data preprocessing method for ATP purposes. Section 3 formulates the problem definition and briefly summarizes typical solutions for PT-ATP. Section 4 presents the experimental setup, the results of our approach, and compares the performance of different solutions, followed by conclusions in Section 5.

2 Preprocessing static and real-time GTFS data

In this section, we present some preliminary terms for the GTFS and GTFS-RT feeds. Also, we describe the employed GTFS data, the GTFS errors and the proposed GTFS data preprocessing method for ATP purposes.

2.1 PT provider & GTFS data

The American Public Transportation Association named Metro Transit is the primary PT operator in the Minneapolis–Saint Paul of the U.S. state of Minnesota and the largest operator in the state. Metro Transit provides an integrated network of buses, light rail, and commuter trains, and has adopted GTFS format to share information with the public. Particularly, the website at svc.metrotransit.org is well maintained with frequent updates of GTFS and GTFS-RT feeds. More specifically, GTFS static data are downloaded as a zip file[1] and are updated weekly, but are subject to change at any time and daily checks are recommended. As far as the GTFS-RT feeds are concerned, they are updated every 15 seconds and include three feeds: the TripUpdate feed[2], the VehiclePosition feed[3] and the ServiceAlerts feed[4].

[1] https://svc.metrotransit.org/mtgtfs/gtfs.zip
[2] https://svc.metrotransit.org/mtgtfs/tripupdates.pb
[3] https://svc.metrotransit.org/mtgtfs/vehiclepositions.pb
[4] https://svc.metrotransit.org/mtgtfs/alerts.pb

In general, a GTFS feed is a collection of at least six comma-separated values (CSV) files (agency, routes, trips, stops, stop times, calendar) and seven optional ones. Metro Transit provides 11 files: agency, routes, trips, stops, stop times, calendar, shapes, calendar dates, feed info, linked datasets, vehicles. In a PT network, stops represent the available stations at which the PT vehicles can stop to pick up or drop off passengers. A sequence of stops constitutes a route. Multiple routes may use the same stop. Each route has a schedule that is followed by a PT vehicle and each route is composed of many trips, which follow the same route, but occur at a specific time throughout a day. Moreover, shapes describe the path that a vehicle travels along a route alignment, are associated with trips, and consist of a sequence of points through which the vehicle passes in order. Stops on a trip should lie within a small distance of the shape for that trip.

As far as the GTFS-RT feed is concerned, it allows PT agencies to provide real-time updates about their fleet through three different types of live feed-trip updates: Trip Update (provide information about predicted arrival/departure times for stops along the operating trips), Vehicle Position (provide information about the locations of the vehicles, e.g. GPS coordinates), and Service Alerts (provide human-readable descriptions regarding disruptions on the network). More specifically, the Metro Transit Trip Update feed includes information about vehicle's timestamp, trip id, route id, direction id, start time, start date, vehicle id, vehicle label, stop sequence, stop id, arrival time, departure time. Note that the arrival/departure times at stops are the predicted ones; at least in Metro Transit, the actual arrival/departure times are not included in the feed. Also, the Metro Transit Vehicle Position includes information about vehicle's timestamp, trip id, route id, direction id, start time, start date, vehicle id, vehicle label, position latitude, position longitude, bearing, odometer, speed. Finally, since the Metro Transit Service Alerts feed includes human-readable descriptions, which are not easily manageable automatically this feed is not used in this work.

2.2 GTFS data errors & proposed solutions

GTFS data often contain errors, such as misrepresentations of the actual network, stops could be encoded imprecisely and have incorrect coordinates. Several GTFS errors for the static counterpart need to be resolved in order to be useful for data analytic purposes, where the most common errors along with the provided solutions, are presented below:

- Duplicate trip information: Each trip should be unique within a route (i.e. same trips, at the same times, should not occur), otherwise is eliminated.
- Incorrect or duplicate stop timestamps: Scheduled arrival and departure times should increase for each stop along the trip and should not be the same at three or more consecutive stops, otherwise the respective trip is eliminated.
- Incorrect stops: Stops coordinates and sequence should match the road network coordinates and direction, respectively, generated from the available shapes. The incorrect stops are eliminated and if a large number of stops within a trip is incorrect, then the whole trip is eliminated.

– Incorrect shapes: Shapes coordinates should match the actual road network coordinates and direction generated from OpenStreetMap [18], otherwise the respective shape is eliminated.

Moreover, GTFS-RT feed contains errors, such as mismatches with the scheduled data, that need to be resolved in order to be useful for data analytic purposes, where the most common errors along with the provided solutions, are presented below:

– Route/Trip/Stop ids mismatching: The vehicle route/trip/stop ids in GTFS-RT should be included in GTFS static feed, otherwise the respective points are eliminated.
– Unrealistic alighting times: Multiple consecutive timestamps for one vehicle position may occur when arrival or departure events occur. These timestamps must occur in short times otherwise the ids are invalid and are eliminated.
– Multiple vehicle positions for one timestamp: For each timestamp only one vehicle position should be recorded, otherwise the most reasonable value is kept.
– Incorrect vehicle position data: Vehicle positions should match the available GTFS shapes (i.e. vehicle positions should be within a buffer surrounding the GTFS shapes) of the trip that the vehicle operates and the road distances between consecutive positions should result in reasonable values, otherwise the points are invalid and are eliminated.
– Invalid timestamps: Timestamps should be strictly sorted for a specific vehicle operating on a specific trip and the time differences between consecutive positions should result in reasonable values, otherwise the timestamps are invalid and are eliminated.
– Invalid vehicle speed: The vehicle speed is calculated by using the road distance between two vehicle positions and the time horizon between the corresponding timestamps. The vehicle speed should follow the minimum and maximum speed limits defined by Metro Transit, otherwise the validity of the associated vehicle positions and timestamps should be investigated.
– Invalid trip start times: The GTFS-RT trip start timestamp should match the scheduled arrival time of the first stop of the trip provided by the "stop times" file, otherwise the trip is invalid and is eliminated.
– Invalid trip start/end times and start/end stop positions: The vehicles may report timestamped positions long time before the scheduled trip start time and/or on a different position of the first stop location of the trip. Also, the vehicles may report timestamped positions many kilometres from the last reported location before going out of service. These are considered extreme errors and are resolved by a) deleting the vehicle's positions with distance from the stop location higher than 50 meters, b) deleting the timestamps that differ from the scheduled trip start time more than a specific amount of time which is equal to the maximum delay time of the trips of the same road id at the first stop, and c) deleting the timestamps that differ from the

scheduled trip end time more than a specific amount of time which is equal to the maximum delay time of the trips of the same road id at the last stop.

2.3 Cleansing and Reconstructing GTFS data (CR-GTFS) tool

GTFS static and real-time data main purpose is to share PT information and are not appropriate for being used by ML methods for data analytic purposes directly. To address this problem and to use GTFS and GTFS-RT feeds effectively for learning ATP purposes, we propose the CR-GTFS tool, which downloads, saves, cleans and reconstructs the GTFS data (both static and real-time).

In this work, GTFS and GTFS-RT feeds are retrieved from the Metro Transit. Particularly, CR-GTFS tool downloads the available schedule data every 2 hours and the real-time feeds every 5 seconds to ensure that all data are collected, and stores them in a PostgreSQL database, which is spatially enable by using PostGIS. Then the process to automatically identify the problematic and missing information, and to reconstruct GTFS and GTFS-RT feeds follows and is described subsequently.

As far as the GTFS static feed is concerned, the respective files are merged according to the following flow: agency is merged with routes (using "agency id" as the key), then merged with trips (using "route id" as the key), then merged with stop times (using "trip id" as the key), then merged with stops (using 'stops id" as the key), then merged with shapes (using "shape id" as the key), then merged with calendar and calendar dates (using "service id" as the key). Subsequently, the GTFS data errors described in section 2.2 are resolved and the final GTFS static dataset is a complete dataset that includes all the available timetable information.

As far as the GTFS-RT data are concerned, the TripUpdate feed is merged with the VehiclePosition feed by using a number of different combination of keys, e.g. a) "trip id" and "vehicle id", or b) "route id", "direction id" and "vehicle id" (in the case of missing/faulty values of "trip id"), or c) "route id", "direction id" and vehicle timestamp occur within the time frame of a scheduled trip with "route id" (in the case of missing/faulty values of "vehicle id"). By merging the VehiclePosition with the TripUpdate and by using various combination of merging keys, the proposed tool fills the missing information concerning the common features: vehicle's timestamp, trip id, route id, direction id, start time, start date, vehicle id and vehicle label. Subsequently, the abovementioned reconstructed GTFS static dataset is merged with the GTFS-RT data by using as keys the "road id", the "direction id" and the "trip id". Then, the GTFS-RT data errors described in section 2.2 are resolved and the resulted dataset is a complete set of information about the transit system.

Due to the fact that the GTFS-RT feed is designed to provide only updates on operating vehicles, each reported vehicle is in service and is operating on a trip and is assigned a unique trip id. The trip ends when a) the assigned vehicle on the specific trip does not operate anymore (i.e. the vehicle does not appear in the GTFS-RT feeds), and/or b) the assigned vehicle on the specific trip reports a different route for consecutive timestamps, or a different direction. For each

completed trip, CR-GTFS tool saves the related information to the database and process it. The resulted dataset is a complete dataset that includes all the available real-time information for each vehicle operating on a trip. However, this dataset does not include the information whether a vehicle actually arrived or departed the stop.

For ATP purposes, we need to know the vehicle's arrival time at a stop. This can be addressed by matching the stops from the schedule data to the GTFS-RT stop times based on the timestamps of the vehicles passing those stops, for each GTFS-RT completed trip. However, this is a challenging task due to a number of reasons such as the fact that sometimes no location is reported near a stop as the vehicle passes the stop quickly. In order to solve this problem, we need to determine which stops were passed and estimate their times of arrival and departure. Thus, the CR-GTFS, for each trip, calculates the distance between the timetable stops locations and the available GTFS-RT vehicle's positions. If the distance between a specific stop location and a vehicle's position falls within the range of 30 meters, then the recorded vehicle timestamp matches the stop. Subsequently, the timestamp for each remaining stop (that has not a matched timestamp) can be estimated by using the road network distance and the speed of the vehicle from the previous and next positions of the specific stop's location.

For the experimental purposes of this study we created a one month GTFS data (March 2021), by using the proposed CR-GTFS tool. Note that both bus and rail services were included.

3 Methodology

3.1 Problem formulation

Using the processed dataset as described above, the problem can be formulated as follows:

- **Given**: An input vector $V = \{v_{t-k}, \ldots, v_{t-1}, \hat{v_t}\}$, where t is the current bus stop and v_{t-k} contains sequential information about passing through stop $t-k$,
- **Predict**: The arrival time or $dT_{t,t+1}$ towards the next bus stop in sequence.

For each of the previous k bus stops, the sequential information gathered is stop identifier "stop id", actual distance $dS_{t-j,t-j+1}$ travelled from previous stop (GPS-based), actual time $dT_{t-j,t-j+1}$ for this transition and estimated (mean) speed, i.e., $dS_{t-j,t-j+1}/dT_{t-j,t-j+1}$.

The reason for including redundant information with speed is that some regression models become simplified and easier to train, as the pair distance-and-time introduces non-linearities in the input compared to distance-and-speed. Furthermore, speed may also be included in the next-step sub-vector $\hat{v_t}$ where it is typically not available ($dT_{t,t+1}$ not realized yet), if instead some globally available estimation of it can be attained from historic data, i.e., the mean time it usually takes to travel between these two bus stops in that specific direction.

For more accurate comparison of the examined models, not such additional estimations were made for next-speed elements and, hence, the next-step sub-vector $\hat{v_t}$ contains only "stop id" and actual distance $dS_{t,t+1}$ that are available at any given t.

Although any N-step look-ahead setup can be used in this core regression task, the choice of one-step look-ahead was based on two reasons. First, the purpose here is to compare the methods in the pure short-term sense, i.e., limit the effects of noise and stationarity shifts caused by exogenous localized factors like unstable road traffic. Second, it is straight-forward to extend the one-step look-ahead approach to any N-step option by iterating the same process multiple times and employing a sliding window that incorporates every new prediction; for robust regressors, the expected N-step look-ahead error of this process is typically bounded by N times the one-step error of the model. Therefore, the one-step look-ahead results are a very good indication of how these models will behave if used iteratively and, most importantly, what is their performance if used continuously in an online fashion, e.g. with streaming data as they become available.

3.2 Machine Learning methods compared

As a performance baseline, two variants of the Linear Regression [17] were employed in this study: (a) the standard method based on the least squares error minimization, (b) the same method but enhanced with the M5 algorithm [26] for attribute selection-elimination. Since the regression task at hand is clearly non-linear, every other regression method with realistically usable application should perform better than this baseline.

A simple model that is often used as density-based estimators is the k nearest neighbour (k-nn) [25] approach, more specifically the Instance Based learners (IBk). In regression problems of high or unknown intrinsic dimensionality, or when the underlying target distribution is suspected to be skewed or clustered, the IBk algorithm is often used as a very good indicator of the expected performance of other, properly trained and robust models. However, like the k-nn, it does not include per-se any attribute-selective process and the distance metric may be negatively affected by a few heavily skewed or correlated dimensions. Moreover, the model itself always requires the complete training dataset or a very extensive representation of it, in order to make each decision during the evaluation phase. This is why these models are often called 'lazy' learners. Nevertheless, as density-based, models, they do not depend on a 'global' functional approximation of the entire manifold, but rather good approximation of arbitrary local neighborhoods of the data. In this study, IBk was included as such a representative of 'lazy' learners based on the k-nn approach, using the Manhattan distance metric and inverse distance as weighting factors within each local neighborhood of k samples.

Focusing on the data space partitioning and implicit attribute selection characteristics of decision tree algorithms, the 'Reduced Error Prunning Tree' or REPtree algorithm [13] was used as a representative of this category. It is a fast

decision tree model that can be used for classification or regressions tasks, similarly to the classic Classification and Regression Tree (CART) algorithms [25]. During training, it builds several trees based on a loss minimization criterion, typically the information gain or reduction of variance with each new node split. Due the sorting of numeric features only once, the overall speed of the training process is improved. Subsequently, the tree is pruned for improving its generalization based on an error function, typically mean squared error (MSE) or minimum absolute error (MAE). Finally, it selects the best-performing tree from all the candidates as the representative model.

As a representative of the ensemble methods [5, 14, 27], Additive Regression was also employed in this study. It is a realization of the boosting approach for regression tasks, i.e., training separate subsequent models upon the residuals (errors) of the previous iterations. The final result is an aggregation of all the trained models, which are typically some weak classifier/regressor, e.g. decision trees. It also includes a shrinkage factor for the learning rate, in order to accomplish smoother trained manifold and avoid over-fitting. In this study, REPtree was used as the base regression model for the ensembles.

Finally, a neural network model was employed as a classic 'universal approximator'. in particular, a multi-layer perceptron (NN-MLP) [25] architecture was used with topologies of one, two and three hidden neurons with softmax activation functions. The main advantages of NN-MLP over many other types of regression models is that the (one or more) hidden layer provides a data space partitioning feature similar to the decision trees and at the same time incorrorate a non-linear aggregation scheme for producing the final output. Instead of the classic back-propagation training, the Broyden–Fletcher–Goldfarb–Shanno (BFGS) [2, 4, 10, 22] optimization algorithm was employed instead, due to its enhanced stability and faster convergence. The BFGS algorithm, used in a wide range of (mostly) unconstrained optimization tasks, is based on directional preconditioning of the descent gradient and it is one order faster than the classic Newton methods. It uses localized curvature information via gradual improvements upon the Hessian matrix of the loss function without the need for matrix inversions or analytical gradient definitions.

4 Experimental Study

The proposed method was formulated and experimentally validated over a real-world PT dataset, which was created by using the proposed CR-GTFS tool as presented in Section 2.

4.1 Experimental protocol - Parameter selection

After the full pre-processing of the raw data, the training and testing datasets were prepared according to the specific problem formulation for the regression task, according to the input-output schema described in section 3.1. In practice,

collected sequential data (stop id, elapsed time, distance, speed) from the previous four bus stops, as well as the available data towards the next stop (stop id, distance), are used as input vector; the arrival time to the next stop is the output, i.e., the conditioned variable in the regression.

For proper evaluation, a 5-fold cross-validation process [25] was employed for all trained models, using exactly the same randomization (seed) in order to avoid any partitioning side-effects. Hence, all performances were evaluated on 80% training and 20% testing splits of the initial dataset, in five iterations, and the final numbers are the mean values over these splits. Due to the significant differences in complexity, the training cycles of the various models ranged from 1-2 seconds (Linear Regression) to more than six hours (NN-MLP).

Regarding model parameterization, several aspects of each model type were taken into account and optimized with intermediate experiments before the final performance assessment, with the exception of Linear Regression which is essentially non-parametric. For IBk, the distance metric (Manhattan) and the weighting factor (inverse distance), as well as the size of the neighborhood between $k = \{5, \ldots, 10\}$, were selected as optimal for this task. Similarly, for REPtree the node splitting criterion was selected to 1e-3 and at least four instances per leaf for the pruning, but with no prior constraint for the expansion depth, in order to accommodate the large dataset size without imposed approximation deficiencies. The REPtree was also used as the base weak learner in the Additive Regression model, used without shrinkage factor and with 10 iterations.

Finally, for NN-MLP the main focus of the optimization was in its topology, i.e., the number and size of the hidden layers employed. As expected, in regression tasks any feature space partitioning beyond a single hidden layer does not provide improvements in the final accuracy, single any subsequent aggregation steps may actually increase, instead of decreasing, the approximation error. In other words, and in contrast to the multiple-layer NNs used in deep learning approaches like with auto-encoders used in classification tasks, a properly designed first hidden layer is more than adequate to address an arbitrary regression tasks. It should be noted that the choice of the size of the single (or first of multiple) hidden layers in NN-MLP regressors can be examined in combination with a preliminary clustering step, in order to make a rough estimation of the level of non-uniformity of the input data space which the model can exploit. In this study, up to three hidden layers were tested in NN-MLP topologies, but the best candidates were those with a single hidden layer of size within a range of $n_L = \{10, \ldots, 50\}$. The lower bound was based on clustering estimations via k-means and EM algorithms [25] that yielded a total of 8-10 clusters, 3-4 of which were mapping 8-10% of the data, i.e., can be considered 'outliers' clusters. The upper bound is mostly constrained by the training time required, as well as by the fact that the increase in accuracy (MAE) versus hidden layer size increases only logarithmically (very slow), as it is explained and illustrated below in section 4.2.

4.2 Results and Discussion

Figure 1 illustrates the distribution of the target for arrival time in the dataset, i.e., true $dT_{t,t+1}$, which, as expected, follows a highly skewed Gaussian or a Generalized Extreme Value (GEV) profile [23], with heavy positive tail. This is due to the fact that the dataset contains a few, very large time differences in specific bus routes, i.e., with very sparse bus stops. In order to test the robustness of the models and their training, it was decided not to remove any such extreme values but instead use it as-is, simulating a real-world requirement of having to produce ATP for any given input vector, including extremes.

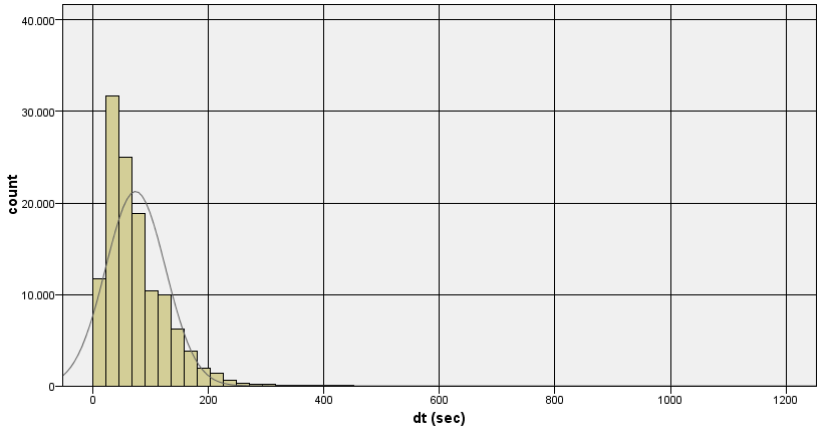

Fig. 1. Distribution of regression target for arrival time in the dataset, i.e., true $dT_{t,t+1}$.

The experimental protocol employed was the same for all models, as described previously in section 4.1. Table 1 presents the results for all the implemented methods, each with its best-performing configuration. For IBk this is with $k = 9$; for NN-MLP this is with a single hidden layer of size $n_L = 50$ (at least $n_L \geq$

Table 1. Results for all the implemented methods

Method	MAE (sec)	RMSE (sec)	R
Linear Regression (std)	29.493	41.021	0.6058
Linear Regression (M5)	26.949	38.487	0.6654
IBk (k-nn)	21.119	31.744	0.7891
REPtree	21.503	32.092	0.7829
Additive Regression	21.427	32.013	0.7842
NN-MLP (hiddenL=1)	**21.050**	**31.243**	**0.7956**
NN-MLP (hiddenL=2)	24.619	35.731	0.7216

35). Bold indicates the overall-best performance given the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE) and Pearson's pairwise correlation coefficient (R) between true and predicted values.

Using the best topology for NN-MLP (single-layer, $n_L = 50$), Figure 2 presents the distribution of errors (MAE) against the regression variable (arrival time). Again, it is obvious that it follows a skewed Gaussian or a Generalized Extreme Value (GEV) profile [23], with moderate positive tail, as expected. However, in contrast to Figure 1, the distribution is much more 'packed' towards zero and the positive tail is suppressed. This essentially means that the resulting NN-MLP 'prefers' to generalize over the main body of the input space, evidently lacking in accuracy on the extreme cases, hence producing the expected prediction error (MAE) somewhat shifted to the right but not very far from zero. In other words, the error profile proves that the NN-MLP exhibits both small prediction error and high level of generalization, biased towards producing larger errors in the extremes rather than throughout the input space.

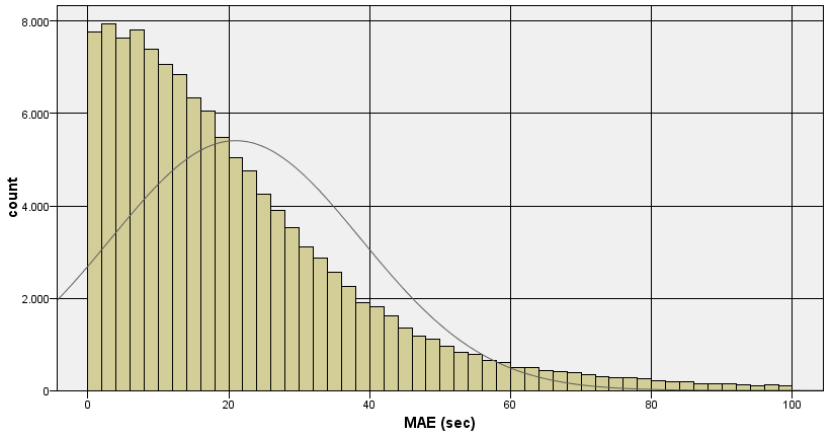

Fig. 2. Distribution of regression error (MAE) for one-step look-ahead ATP using NN-MLP.

Regarding the single-layer NN-MLP, which is the overall-best regressor in this task, it is worth noting that as a model it exhibits a very high level of information 'packing' in its trained parameters: the specific topology of $n_L = 50$ translates to a total of 1,251 weight parameters, including one weight per input attribute ($n_{inp} = 23$) plus one bias coefficient per neuron, i.e., $|W| = n_L(n_{inp}+1)+(n_L+1)$. In contrast, the second-best model which is IBk requires the complete dataset used for k-nn lookups with $k = 9$, a process that is much slower and two orders of magnitude more space-demanding. Similarly, single and ensemble REPtree (Additive Regression) comes close in terms of accuracy, but again the space complexity (tree sizes) is at least 4-5 times larger than the NN-MLP model. Furthermore, the ensemble option (multiple trees) are usually required in order

to cope with the inherent noise sensitivity (instability) of single decision trees and the improvement of generalization.

Regarding the trade-off of the size of the single hidden layer in NN-MLP versus the performance improvement, Figure 3 illustrates some reference points and the corresponding trend in terms of logarithmic fit. The exact formula of the fit is: $MAE \approx f(n_L) = \alpha \cdot \ln(n_L - 8) + \beta$, where $\alpha = -0.640232$ and $\beta = +23.427965$. It is obvious that the trend fit is good, very close to the actual reference points, and the performance gain beyond $n_L \geq 35$ becomes negligible. Nevertheless, even with $n_L = 50$ the NN-MLP topology translates to a model several times smaller in size than the next best alternative.

In summary, the single-layer NN-MLP model outperforms all the other techniques. Besides Linear Regression and two-layer NN-MLP, the performances of the rest of the models differ only marginally; however, due to the very large number of training samples (122,320 in total), the relative ranking of the tested models can be considered as statistically significant and valid for performance comparison. Finally, it should be noted that an increase in the number of samples and the problem dimensionality results in higher computational times in NN-MLP models. However, the performance of the employed NN-MLP model indicates that the algorithm can handle GTFS datasets with large amounts of samples and take advantage of the high prediction accuracy, in contrast to the non NN-based methods, which provide less accurate predictions.

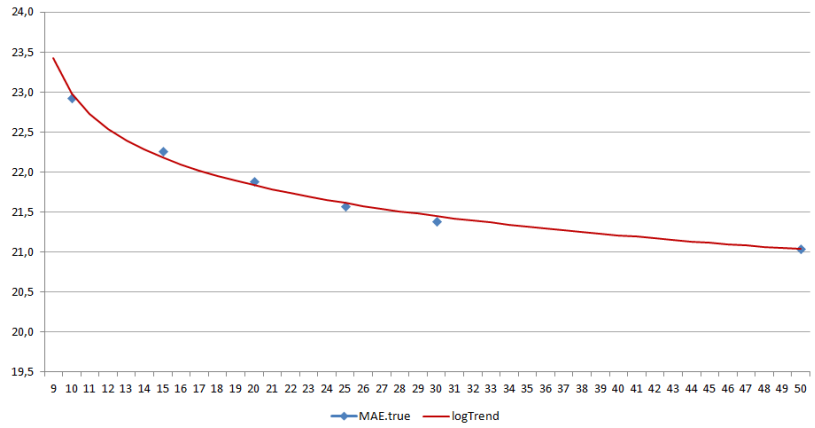

Fig. 3. NN-MLP topology (hidden layer size) versus one-step look-ahead ATP error (MAE); blue dots are real test points (training results) and red line is the estimated trend (logarithmic fit).

5 Conclusion

Due to recent advances in position broadcasting technology and the adoption of a common transit feed format by thousands of PT agencies, PT movement in-

formation has become increasingly available. An effective estimation of PT-ATP is substantial for improving the quality and the reliability of the PT services. Taking advantage of the GTFS data, this work proposes a new framework to address the PT-ATP problem. Also, various ML models are tested in solving the PT-ATP problem. As a result, insightful findings through the comparison procedure are provided. The results showed that the NN-based method outperforms its rivals in terms of prediction accuracy.

Future work includes the investigation of weather information impact on the PT-ATP problem. Also, we plan to experiment with further ML algorithms, such as recurrent NN architectures, taking into account the training computational times. Finally, we plan to focus on a larger prediction time horizon, as well as extending the prediction's window length by including more stops.

Acknowledgements. This paper is one of the deliverables of the project with MIS 5050503, of the Call entitled "Support for researchers with emphasis on young researchers - cycle B" (Code: EDBM103) which is part of the Operational Program "Human Resources Development, Education and Lifelong Learning", which is co-financed by Greece and the European Union (European Social Fund).

References

1. Alam, O., Kush, A., Emami, A., Pouladzadeh, P.: Predicting irregularities in arrival times for transit buses with recurrent neural networks using gps coordinates and weather data. Journal of Ambient Intelligence and Humanized Computing pp. 1–14 (2020)
2. Broyden, G.: The convergence of a class of double-rank minimization algorithms. Journal of the Institute of Mathematics and Its Applications **6**, 76–90 (1970)
3. Caiati, V., Bedogni, L., Bononi, L., Ferrero, F., Fiore, M., Vesco, A.: Estimating urban mobility with open data: A case study in bologna. In: 2016 IEEE International Smart Cities Conference (ISC2). pp. 1–8 (2016)
4. Fletcher, R.: A new approach to variable metric algorithms. Computer Journal **13**(3), 317–322 (1970)
5. Friedman, J.: Stochastic gradient boosting. Tech. rep., Stanford University (1999)
6. Georgiou, H., Karagiorgou, S., Kontoulis, Y., Pelekis, N., Petrou, P., Scarlatti, D., Theodoridis, Y.: Moving Objects Analytics: Survey on Future Location & Trajectory Prediction Methods. arXiv e-prints arXiv:1807.04639 (2018)
7. Georgiou, H., Pelekis, N., Sideridis, S., Scarlatti, D., Theodoridis, Y.: Semantic-aware aircraft trajectory prediction using flight plans. International Journal of Data Science and Analytics **9**(2), 215–228 (2020)
8. Georgiou, H., Petrou, P., Tampakis, P., Sideridis, S., Chondrodima, E., Pelekis, N., Theodoridis, Y.: Future Location and Trajectory Prediction, pp. 215–254. Springer International Publishing, Cham (2020)
9. Ghanim, M., Shaaban, K., Miqdad, M.: An artificial intelligence approach to estimate travel time along public transportation bus lines. In: The International Conference on Civil Infrastructure and Construction. pp. 588–595 (02 2020)
10. Goldfarb, D.: A family of variable metric updates derived by variational means. Mathematics of Computation **24**(109), 23–26 (1970)

11. Google: Testing gtfs feeds. https://developers.google.com/transit/gtfs/guides/tools (June 2021)
12. Hua, X., Wang, W., Wang, Y., Ren, M.: Bus arrival time prediction using mixed multi-route arrival time data at previous stop. Transport **33**(2), 543–554 (2018)
13. Kalmegh, S.: Analysis of weka data mining algorithm reptree, simple cart and randomtree for classification of indian news. International Journal of Innovative Science Engineering and Technology **2**(2), 438–446 (2015)
14. Kuncheva, L.: Combining Pattern Classifiers: Methods and Algorithms. Wiley-Interscience, USA (2004)
15. Larsen, G.H., Yoshioka, L.R., Marte, C.L.: Bus travel times prediction based on real-time traffic data forecast using artificial neural networks. In: 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE). pp. 1–6 (2020)
16. Liu, H., Xu, H., Yan, Y., Cai, Z., Sun, T., Li, W.: Bus arrival time prediction based on lstm and spatial-temporal feature vector. IEEE Access **8**, 11917–11929 (2020)
17. Montgomery, D., Runger, G.: Applied Statistics and Probability for Engineers (7th/Ed.). John Wiley & Sons (2018)
18. OpenStreetMap contributors: Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org (2017)
19. Petersen, N.C., Rodrigues, F., Pereira, F.C.: Multi-output bus travel time prediction with convolutional lstm neural network. Expert Systems with Applications **120**, 426 – 435 (2019)
20. Petrou, P., Tampakis, P., Georgiou, H., Pelekis, N., Theodoridis, Y.: Online long-term trajectory prediction based on mined route patterns. In: International Workshop on Multiple-Aspect Analysis of Semantic Trajectories. pp. 34–49. Springer, Cham (2019)
21. Ranjitkar, P., Tey, L.S., Chakravorty, E., Hurley, K.L.: Bus arrival time modeling based on auckland data. Transportation Research Record **2673**(6), 1–9 (2019)
22. Shanno, D.: Conditioning of quasi-newton methods for function minimization. Mathematics of Computation **24**(111), 647–656 (1970)
23. Spiegel, M., Schiller, J., Srinivasan, R.: Probability and Statistics (3rd/Ed.). McGraw-Hill (2009)
24. Sun, F., Pan, Y., White, J., Dubey, A.: Real-time and predictive analytics for smart public transportation decision support system. In: 2016 IEEE International Conference on Smart Computing (SMARTCOMP). pp. 1–8 (2016)
25. Theodoridis, S., Koutroumbas, K.: Pattern Recognition. Academic Press, 4th edn. (November 2008)
26. Witten, I., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann Publishers, USA (2000)
27. Yuksel, S., Wilson, J., Gader, P.: Twenty years of mixture of experts. IEEE Trans. on Neural Networks **23**(8), 1177–1192 (August 2012)
28. Čelan, M., Lep, M.: Bus arrival time prediction based on network model. Procedia Computer Science **113**, 138 – 145 (2017), the 8th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2017) / The 7th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2017) / Affiliated Workshops