CANADA EXCELLENCE RESEARCH CHAIR

DATA SCIENCE FOR REAL-TIME DECISION-MAKING

TRAVEL SPEED PREDICTION BASED ON LEARNING METHODS FOR HOME DELIVERY

Maha Gmira
Michel Gendreau
Andrea Lodi
Jean-Yves Potvin

November 2018

DS4DM-2018-012

POLYTECHNIQUE MONTRÉAL
DÉPARTEMENT DE MATHÉMATIQUES ET GÉNIE INDUSTRIEL
Pavillon André-Aisenstadt
Succursale Centre-Ville C.P. 6079
Montréal - Québec
H3C 3A7 - Canada
Téléphone: 514-340-5121 # 3314

Travel Speed Prediction based on Learning Methods for Home Delivery

Maha Gmira †§¶, Michel Gendreau †¶
Andrea Lodi †§¶, Jean-Yves Potvin ‡¶

†Département de mathématiques et de génie industriel
Polytechnique Montréal,
C.P. 6079, succ. Centre-Ville, Montréal, Québec, Canada H3C 3A7.

‡Département d'informatique et de recherche opérationnelle
Université de Montréal,
C.P. 6128, succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7.

§Chaire d'excellence en recherche du Canada sur la science des données
pour la prise de décision en temps réel
Polytechnique Montréal,
C.P. 6079, succ. Centre-Ville, Montréal, Québec, Canada H3C 3A7.

¶Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport,
Université de Montréal,
C.P. 6128, succ. Centre-Ville, Montréal, Québec, Canada H3C 3J7.

Abstract. The travel time required to get from one location to another in a network is an important performance measure in intelligent transportation and advanced traveler information systems. Accordingly, accurate travel time predictions are of foremost importance. In an urban environment, vehicle speed and consequently travel time can be highly variable due to congestion caused, for example, by accidents or bad weather conditions. At another level, one also observes daily patterns (e.g., rush hours),

weekly patterns (e.g., weekdays versus weekend), and seasonal patterns. Capturing time-varying features when modeling travel speeds provide an immediate benefit to commercial transportation companies that distribute goods, since it allows them to better optimize their routes and reduce their environmental footprint.

This paper presents a travel speed prediction methodology based on data collected from mobile location devices installed inside commercial delivery vehicles. An analysis is conducted using unsupervised learning to cluster data, dimensionality reduction techniques and imputation methods to fill missing values and a Long Short-Term Memory neural network to forecast travel speeds.

1 Introduction

Traffic congestion can be classified as recurrent (due to well-known patterns) and non-recurrent (due to accidents, construction, emergencies, special events and bad weather, among others). Accordingly, there is a need for models that can derive future values from observed trends, in order to provide accurate predictions under recurrent and non-recurrent congestion. With the increasing amount of available data collected from probe vehicles, smartphone's applications and other location technologies, the challenge is no longer related to the quantity of data but rather to the modeling and extraction of useful information from such data. This information can be of great value for transportation companies operating in urban areas.

Given the variability of travel speeds due to usual and unusual traffic situations, the objective of this research is to make better travel speed predictions, which should ultimately help to generate better vehicle delivery routes. This will be done with a neural network model that uses information extracted from data collected from mobile location devices. The data for this study come from a software development company located in Montreal that produces vehicle routing algorithms to plan the home delivery of large items (appliances, furniture) to customers. This partner has delivery routes with more than 2,500,000 delivery points, serviced by nearly 200,000 routes. Data are collected using Automatic Vehicle Location (AVL) systems where GPS receivers are usually interfaced with Global System for Mobile Communications (GSM) modems. The system records point locations as latitude-longitude pairs, instantaneous speed, date and time. Our challenge here was to develop techniques for the management of big data that would allow prediction algorithms to perform well.

In the following, a review of the scientific literature related to our work is first presented in Section 2. Then, our methodology for travel speed prediction is reported. The creation of a database of speed patterns from GPS traces is described in Section 3. Then, techniques to reduce the size of the database and cluster arcs into similarity classes are explained in Section 4. This is followed by the neural network model used

to predict travel speeds in Section 5. Computational results are finally reported in Section 6.

2 Literature review

Most urban traffic control systems rely on short-term traffic prediction and a huge literature has been published on this topic in the last decades due, in particular, to the advent of Intelligent Transportation Systems (ITS). Given that these systems are highly dependent on accurate traffic information, they must collect a large amount of data (locations, speeds and individual itineraries).

Travel speed prediction at a given time typically relies on historical travel speed values and a set of exogenous variables. Methods to predict traffic information are classified in [49] as 1) naive (i.e., without any model assumption), 2) parametric, 3) non-parametric and 4) a combination of the last two, called hybrid methods. The first three methods are described in the following.

2.1 Naive methods

Naive methods are by far the easiest to implement and to use because they do not require an underlying model. However, one main drawback is their lack of accuracy. In [49], naive methods are divided into instantaneous methods (based on data available at the instant the prediction is performed), historical methods (based on historical data) and mixed methods, where the latter combine characteristics of historical and instantaneous methods. As a baseline for comparison with other parametric methods, the work reported in [44] uses a simple hybrid method where the travel speed forecast is a function of the current traffic flow rate, as well as its historical average at a given time of the day and day of the week. In [57], the authors compare two methods for travel time prediction, one based on data available at the instant the prediction is performed and one based on historical data. Using the Relative Mean Error (RME) and the Root Mean Squared Error (RMSE), these two approaches showed similar performance, but were clearly outperformed by a more sophisticated approach called Support Vector Regression (see Section 2.3).

2.2 Parametric methods

Parametric methods use data to estimate the parameters of a model, whose structure is predetermined. The most basic model is linear regression, where the traffic variable V_t to be predicted at a given time t is a linear function of independent variables:

$$V_t = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n \ . \tag{1}$$

Different techniques are available to estimate the parameters β_i, $i = 1, \ldots n$. Among parametric methods, we consider the Autoregressive Moving Average model (ARMA) and its Integrated variant (ARIMA), different smoothing techniques and the Kalman filter. They are presented below.

(a) Kalman Filter

The Kalman filter is a very popular short-term traffic flow prediction method. It allows the state variables to be updated continuously, which is very important in time-dependent contexts. Some works related to traffic state estimation are based on the original Kalman filter [28], as well as its extension for non-linear systems called the Extended Kalman Filter (EKF) [27]. The latter is particularly relevant since the travel times depend on traffic conditions that are highly non-linear and dynamic, changing over time and space. In [55], a freeway state estimator is obtained by solving a macroscopic traffic flow model with EKF. A new EKF based on online-learning is used in [50] to provide travel time information on freeways. Also, a dynamic traffic assignment model, which is a non-linear state-space model, is solved by applying three different extensions of the Kalman filter [1].

In [26], the authors use traffic data on California highways to predict travel times on arcs and estimate the arrival time at a destination. First, travel times on arcs are predicted by feeding the Kalman filter with historical data. Then, this prediction is corrected and updated with real time information using the Kalman filter's corrector-predictor form.

(b) ARMA

To predict short-term traffic characteristics such as speed, flow or travel time, time series models based on ARMA(p,q) (which is a combination of p autoregressive terms AR and q moving average terms MA) have been widely used. If we consider travel time prediction, the general formulation of ARMA(p,q) is:

$$T(t) - \sum_{i=1}^{p} \alpha_i T(t-i) = Z(t) + \sum_{j=1}^{q} \beta_j Z(t-j) \ , \tag{2}$$

where the travel time $T(t)$, given departure time t, is a linear function of the travel times at previous instants, $Z(t-1), \ldots, Z(t-q)$ are noise variables and (α_i, β_j) are parameters.

ARIMA models generalize ARMA models for non-stationary time series. They rely on stochastic system theory since the processes are non-deterministic. In [22], the authors use the Box-Jenkins approach [4] to develop a forecasting model

based on ARIMA from data collected in urban streets (i.e., 1-minute traffic-volume on each street during peak periods). After comparing several ARIMA models, the one of order (0,1,1) yielded the best results in terms of traffic volume forecasts, where the values of 0, 1 and 1 refer to the order of the autoregressive model (number of time lags), number of differentiation steps (number of times the values in the past are subtracted) and moving-average terms, respectively. The authors in [44] compare a seasonal ARIMA model, called SARIMA, with a non-parametric regression model where the forecast generation method and neighbor selection criteria are heuristically improved. The tests showed that SARIMA performed better than the improved non-parametric regression.

In [56], the authors model traffic flow with SARIMA (1,0,1) and SARIMA(0,1,1) models. Another SARIMA model is reported in [20] to forecast traffic conditions over a short-term horizon, based on 15-minute traffic flow data. In this case, SARIMA outperformed the K-nearest neighbor method.

2.3 Non-parametric methods

Non-parametric methods include non-parametric regression and different types of neural networks. Non-parametric methods are also known as data-driven methods, because they need data to determine not only the parameters but also the model structure. Thus, the number and types of parameters are unknown a priori. The main advantage of these methods is that they do not require expertise in traffic theory, although they need a lot of data.

The most popular non-parametric methods for traffic prediction are the support vector machine, neural networks and non-parametric regression.

(a) Support Vector Machine

The Support Vector Machine (SVM) was first introduced in [51, 52] and used in many classification and regression studies. SVM is popular because it guarantees global minima, it deals quite well with corrupted data and works for complex non-linear systems. Support Vector Regression (SVR) is the application of SVM to time-series forecasting.

In [57], the authors analyze the application of SVR for travel time prediction. They used traffic data, obtained from an Intelligent Transportation System, over a five weeks period. The first four weeks correspond to the training set and the last week corresponds to the testing set. Using a Gaussian kernel function and a standard SVR implementation, their method improved the RME and RMSE when compared to instantaneous and historical travel time prediction methods.

Due to its promising results, SVR has been used to predict traffic parameters such as traffic flow or travel time. However, the classical SVR, like the one used in [57], cannot be applied in real time because it requires a complete model training each time a new record (data) is added. There are also some variants of SVM for traffic prediction. In [54], the authors report a hybrid model called the chaos-wavelet analysis SVM that overcomes the need to choose a suitable kernel function. In the context of traffic flow prediction for a large-scale road network [58], SVM parameters are optimized by a parallel genetic algorithm, thus yielding a Genetic Algorithm-Support Vector Machine (referred to as GA-SVM).

(b) Neural networks

When considering data-driven methods, neural networks are among the best for traffic forecasting because of their ability to learn non-linear relationships among different features without any prior assumption.

The most widely used neural networks are called Multi-Layer Perceptrons (MLPs). They are typically made of an input layer, one hidden layer and an output layer, where each layer contains one or more units (neurons). The units in the input layer are connected to those in the hidden layer, while the units in the hidden layer are connected to those in the output layer. The weights on these connections are adjusted during the learning process using (input, target output) example pairs, so as to produce an appropriate mapping between the inputs and the target outputs. In [8], a MLP is used to predict traffic flow from input data (speed, flow, occupancy) collected by detection devices on a highway around the city of London. In that application, the MLP and a radial basis function network performed better than all ARIMA models considered. In [53], the authors exploit a genetic algorithm to fine tune the parameters and the number of hidden units in a MLP. Their model showed better generalization abilities when tested with new inputs. More recently, the authors in [21] report the performance of a MLP with 15 hidden units, trained with the Levenberg-Marquardt backpropagation algorithm. Based on different error performance indicators, the MLP showed good accuracy when predicting road speeds. In [34], a MLP is used to predict the time needed to cross the Ambassador bridge, one of the busiest bridges at the Canada-US border. A database of GPS records for a full year was used to train and test the neural network.

As indicated in [32], Recurrent Neural Networks (RNNs) are better suited for traffic forecasting tasks due to their ability to account for sequential time-dependent data. Typically, the signal sent by the hidden layer to the output layer at some time t is also sent back to the hidden layer. This signal is pro-

cessed with the input signal at time $t + 1$ to determine the internal state of the hidden layer. This internal state acts as a memory and remembers useful time-dependent relationships among data.

A specific class of RNNs, called Long Short-Term memories (LSTM), is now widely used in the literature due to its proven ability to learn long-term relationships in the data. In [32], LSTM is compared to various RNNs and other statistical methods, namely: Elman neural network, non-linear autoregressive with exogenous inputs (NARX) neural network, Time-Delay Neural Network (TDNN), SVM, ARIMA and Kalman filter. The LSTM achieved the best performances in terms of accuracy and stability. The work in [48] proposes a LSTM model for short-term traffic flow prediction and compared it with Random Walk (RW), SVM, MLP and Stacked Autoencoder (SAE). The results showed the superiority of LSTM in terms of prediction accuracy, ability to memorize long-term historical data and generalization capabilities. In [15], LSTM and a neural network model made of Gated Recurrent Units (GRUs) [9] are applied to traffic data in California. The study showed that GRUs behave slightly better than LSTM, while both outperformed an ARIMA model in terms of Mean Squared Error (MSE) and Mean Absolute Error (MAE).

(c) Non-parametric regression

Another class of non-parametric methods is called Non-Parametric Regression (NPR). It is suitable for short-term traffic prediction since it can deal with the uncertainty in traffic flows. The objective of NPR is to estimate a regression function without relying on an explicit form, unlike the traditional parametric regression models (where the model parameters are estimated). The forecasting ability of NPR relies on a database of past observations. It applies a search procedure to find observations in this database that are similar to the current conditions. Then, it transfers these observations to the forecast function to estimate the future state of the system.

The k-Nearest Neighbor (k-NN) is a widespread class of non-parametric regression methods made of two components:

(i) Search procedure: the nearest neighbors (historical data most similar to the current input) are the inputs to the forecast function aimed at generating an estimate. The nearest neighbors are found using a similarity measure, which is usually based on the Minkowski distance metric:

$$L_r = \left(\sum_{i=1}^{n} |p_i - q_i|^r \right)^{1/r} , \tag{3}$$

where n is the dimension of the state vector, p_i is the i^{th} element of the historical record currently considered, q_i is the i^{th} element of the current conditions, and r is a parameter with values between 1 and 2. The most common implementation uses a sequential search procedure. However, as the number of historical observations increases, the sequential search becomes very time consuming.

(ii) Forecast function: the most general approach to generate a prediction is to compute an average of the dependent variable values over the nearest neighbors. However, this approach ignores the information provided by the distance metric (i.e., past observations closer to the current input should have more impact on the forecast).

The authors in [10] were among the first to use NPR to estimate short-term traffic flows. Their work highlighted the importance of a large and representative dataset. NPR was then applied to estimate traffic volumes from two sites located in Northern Virginia Capital Beltway, based on five months of observations [42]. The results showed that the proposed method can generate more accurate predictions than the other tested approaches (including a neural network model). The work in [43] compares historical averages, time series, back-propagation neural networks and non-parametric regression models using a performance index that includes absolute error, error distribution, ease of model implementation and model portability. Overall, NPR proved to be better than the other models and was also easier to implement.

The interested reader is referred to [13] for a more exhaustive review on traffic forecasting. The following sections will now describe the various steps that we performed to produce travel speed predictions from our huge database.

3 Deriving speed patterns from GPS traces

Our industrial partner provided us with one year and a half of GPS data transmitted by mobile devices installed in delivery vehicles (extending over the years 2013, 2014 and 2015). These GPS points were first mapped to the underlying Montreal Metropolitan Community (MMC) network to generate daily speed patterns for each individual arc, where a daily speed pattern for a given arc is made of 96 average speeds taken over time intervals of 15 minutes.

In the following, the main issues related to the derivation of speed patterns from GPS traces are briefly discussed.

3.1 Data preparation

The record of each GPS point contains an identifier, a latitude-longitude pair, an instantaneous speed, a mobile identifier, a driver identifier, a date and a time stamp (Figure 1).

record identifier	latitude	longitude	speed	mobile identifier	driver identifier	date time

Figure 1 – Record structure of a GPS point

The available data had first to be cleaned by deleting GPS points with aberrant speeds (values less than 0 km/h or greater than 150 km/h), aggregates of GPS points associated, for example, with parking stops for deliveries, etc. Then, the map-matching algorithm was applied to the remaining GPS points, as described below.

3.2 Map-matching algorithm

Due to the relatively low accuracy of GPS systems, assigning a GPS point to an arc of the underlying network is a difficult problem, particularly in dense urban road networks. Thus, a good map-matching algorithm is required. To this end, we used a recent algorithm reported in [23] which was slightly adapted to our context. The algorithm works as follows:

Step 1. Identification of trips. A total of 170 and 327 different vehicle and driver identifiers were found over all GPS points. Within a single day, it is possible to find one driver associated with one vehicle, one driver associated with two vehicles or more, and two drivers or more associated with one vehicle. Thus, there are clearly different trips within a day, where a trip corresponds to a vehicle/driver pair.

Step 2. For each trip, all GPS points associated with it are considered for assignment to arcs of the network. This is done in three main phases:

2.1 In the initialization phase, candidate arcs are those that are adjacent to the three nearest nodes of the first GPS point. A score is calculated for each arc based on their closeness to the GPS point, and the arc with the best score is selected. Then, a confidence level is calculated for the selected arc to account for the uncertainty of that choice (due to inherent uncertainty in positioning sensors and digital maps). Here, the confidence level is based on the difference between the score of the selected arc and the second best score. Clearly, a larger difference implies a higher confidence level. If the confidence level is above a

given threshold, the GPS point is assigned to the selected arc, otherwise it is skipped and the next GPS point is considered. Once a GPS point is assigned to the first arc, the same-arc phase starts.

2.2 In the same-arc phase, the next GPS point is assigned to the same arc than the previous one, unless some conditions are not satisfied anymore (e.g., when crossing an intersection). In the latter case, the algorithm switches to the next-arc phase.

2.3 In the next-arc phase, candidate arcs are those connected to the previously selected arc plus those that are adjacent to the three nearest nodes of the current GPS point. A score is calculated for each arc based on its closeness to the GPS point and the difference in direction between the arc and the line connecting the previous GPS point to the current one. Again, the arc with the best score is selected, its confidence level is calculated, and topological constraints are checked (e.g., arc not connected to the previous one, turn restrictions, etc.). Depending on the confidence level and satisfaction of the topological constraints, the current GPS point can either be skipped or assigned to the new arc. In the latter case, the algorithm returns to the same-arc phase. It should be noted that considering arcs that are close to the current GPS point, but not necessarily connected to the previous arc, allows the algorithm to account for discontinuities in GPS traces due to obstacles (e.g., tunnels, bridges).

The interested reader is referred to [23] for details about this algorithm.

When the assignment of GSP points to arcs is completed for each day, average travel speeds can be calculated over time slots of 15 minutes. Thus, a new database is obtained where each record has the structure shown in Figure 2. The next section will now explain how this database was exploited to fit our purposes.

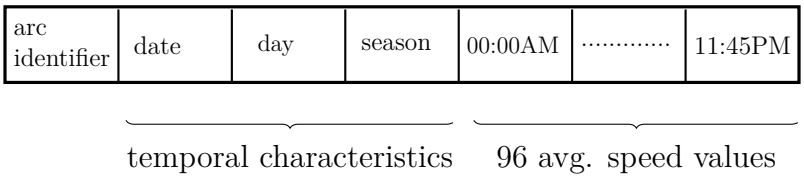

Figure 2 – Database structure after geomatic analysis

4 Size reduction and clustering

Due to the size of our database, size reduction techniques were applied. After eliminating speed patterns and hours with too many missing data, a "prediction-after-classification" approach was used to cluster arcs with similar speed patterns into classes before predicting travel speed values. This is explained in the following.

4.1 Database reduction

With 233,914 arcs and 515 days in the database, we have a total of 233,914 $\times$ 515 = 120,506,910 speed patterns. Originally, the database was constructed with time intervals of 15 minutes. That is, a speed pattern for a given arc on a given day is made of 96 average speed values taken over time intervals of 15 minutes, thus covering an horizon of 24 hours. Furthermore, the speed limit was stored when no observation was recorded within a 15-minute time interval.

An elimination procedure was first applied to get rid of speed patterns or time intervals with too few data, as described below.

(a) *Speed patterns.* To keep only average speed values based on real observations, the speed limit values were removed from the database. Given that a significant proportion of the resulting speed patterns now contained missing data, a speed pattern was automatically discarded when the proportion of real average speed values over the 96 time intervals was less than 5% (note that this threshold is often suggested in the literature). In other words, a speed pattern with only 4 average speeds or less was eliminated. Through this process, we ended up with a total of 6,667,459 speed patterns. It should be noted that only 3,485 arcs still had at least one representative speed pattern in this database. The fact that the original GPS traces have been collected from delivery routes in particular sectors of an urban area explains this large reduction in the number of arcs and speed patterns. Finally, the 96 time intervals of 15 minutes of each remaining speed pattern were aggregated into 24 one-hour time intervals, to allow the calculation of average speed values based on more observations in each time interval, see Figures 3 and 4.

(b) *One-hour time intervals.* After reducing the number of speed patterns in the database, as well as the number of average speed values stored in a pattern, we then examined more closely the one-hour time intervals.

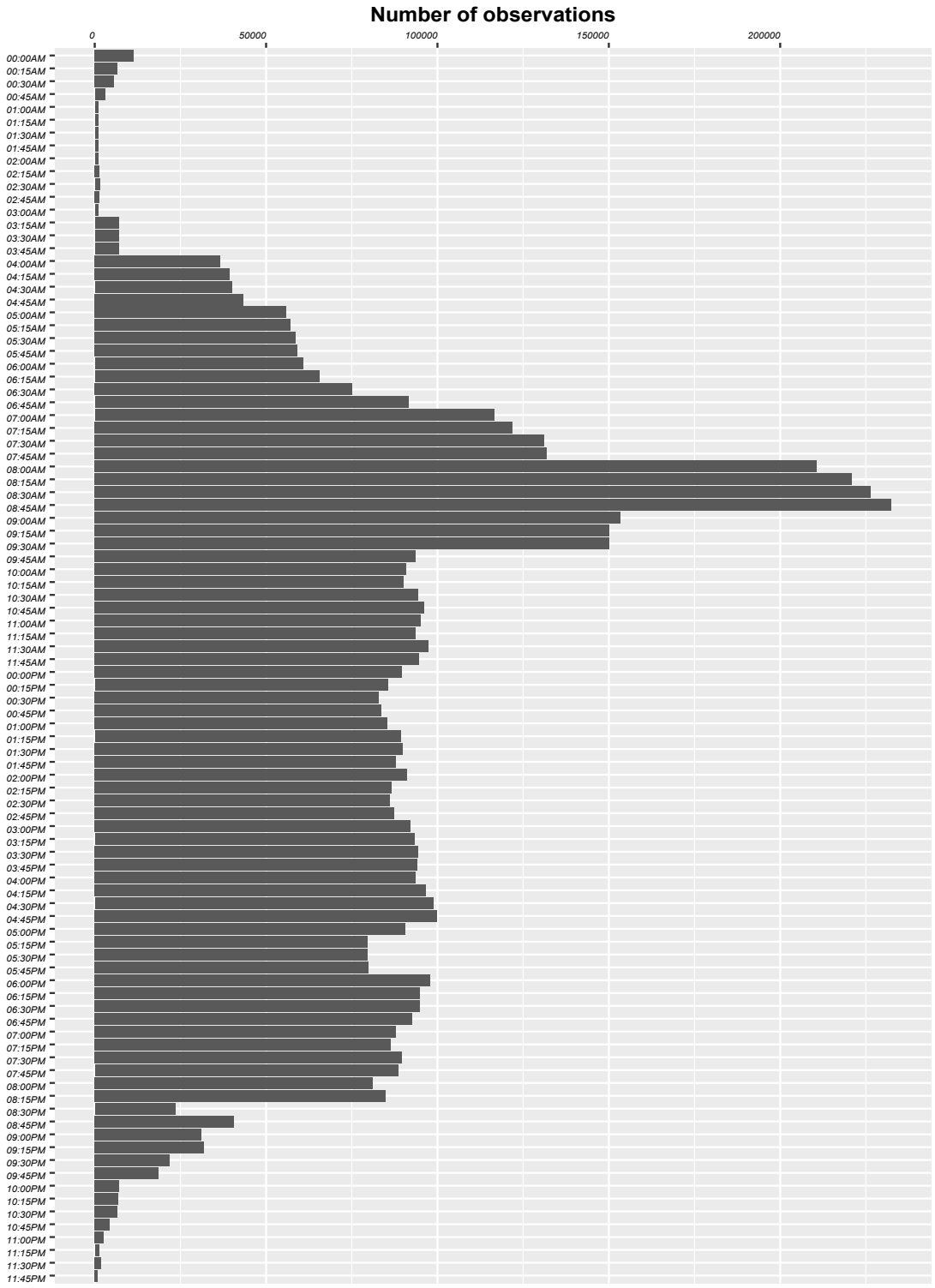

Figure 3 – Number of observations per time interval of 15 minutes

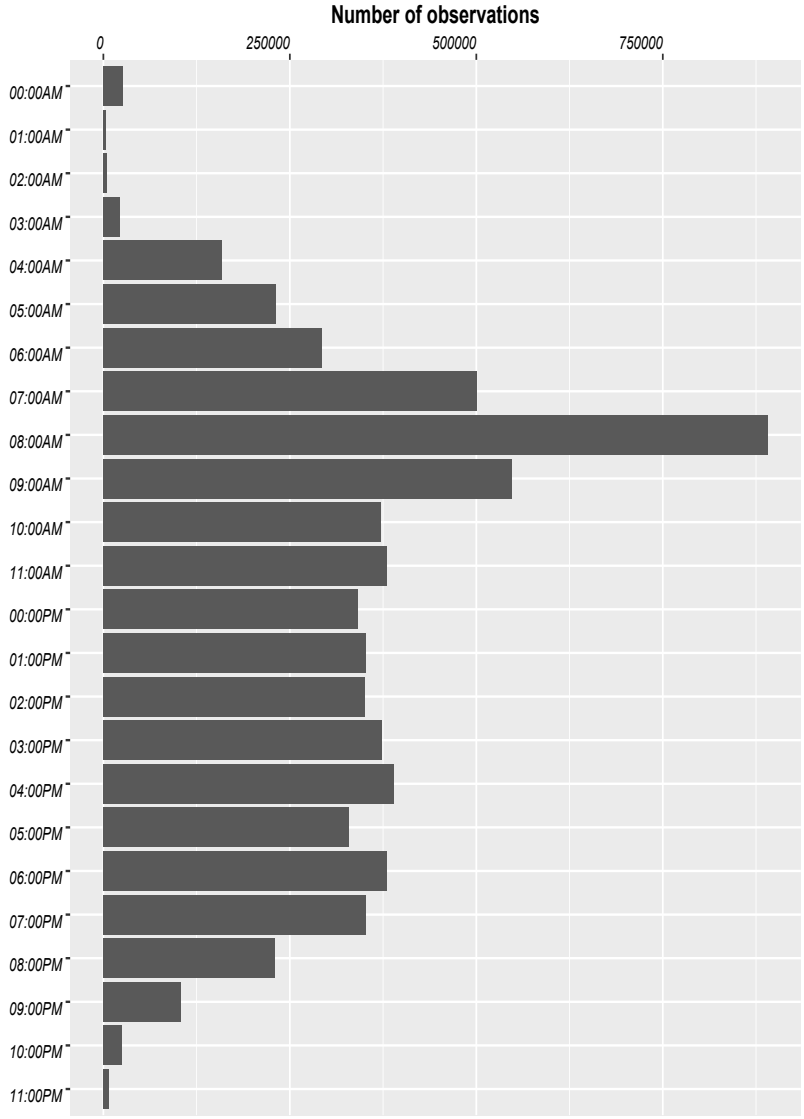

Figure 4 – Number of observations per time interval of 1 hour

Clearly, there are intervals with no or very few observations over the entire database, like night hours (although some observations can be found in the database because the vehicles are sometimes moved during the night from one location to another). Accordingly, we discarded hours where the proportion of real average speed values over the 6,667,459 speed patterns in the database was under 5%. After this elimination process, the number of one-hour time intervals was reduced from 24 to 13. More precisely, only one-hour time intervals starting from 7:00 AM to 7:00 PM were kept in every speed pattern.

4.2 Clustering

When this step is reached, we have a database of 6,667,459 speed patterns, where each pattern is associated with a given arc and a given date. A speed pattern can be seen as a vector of 13 average speed values, one for each one-hour time interval starting from 7:00 AM to 7:00 PM. This number of speed patterns is still too large to be processed by a learning-based prediction algorithm, so we had to group arcs with similar patterns into a number of classes. For this purpose, an average speed pattern was calculated for each arc over all its corresponding speed patterns in the database. Since 3,485 arcs are represented in the database, this led to $N = 3{,}485$ average speed patterns.

In the following, the clustering methods used to identify classes of arcs are described.

4.2.1 *K*-means

To cluster arcs based on their average speed pattern with the K-means algorithm, the distance between two patterns was calculated using the Euclidean metric (see, for example, [33]). At the end, K cluster centroids (classes) were obtained and the speed pattern of each arc was assigned to the closest centroid. Since the number of classes must be fixed in advance, the latter was purposely set to a large value, i.e., $K = 200$. Then, the output of the K-means algorithm was fed to a hierarchical clustering method to further reduce the number of classes (see Section 4.2.2).

The problem solved by the K-means algorithm is summarized below, where C_k stands for class k. The objective is to minimize the sum of the squares of the Euclidean distance between speed pattern x_i of arc i and its closest centroid μ_k, over all arcs. In this objective, the variables are the μ_k's.

$$Min \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2 \ , \tag{4}$$

where:

$$C_k = \{x_i : \|x_i - \mu_k\| = \min_{l=1,\ldots,K} \|x_i - \mu_l\|\} \ , \tag{5}$$

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i \ . \tag{6}$$

An iterative method known as Lloyd's algorithm was used to converge to a local minimum, given that solving the problem exactly is NP-hard. In this algorithm, starting from K randomly located centroids, the following two steps are performed repeatedly until convergence is observed: 1) Cluster assignment: construct the set of classes by assigning each arc to the cluster centroid that is closest to the arc's speed pattern and 2) Update centroids: update the centroid of each cluster by averaging over all speed patterns assigned to it.

4.2.2 Affinity propagation algorithm

The Affinity Propagation (AP) algorithm is a clustering procedure proposed in [14]. As opposed to K-means, every data point is considered as a potential centroid. Through the propagation of so-called affinity values among pairs of data points, which reflect the current affinity (or consent) of one data point to consider the other data point as its centroid, some data points accumulate evidence to be centroids. The reader will find in [14] the exact mathematical formulas that are used to guide the transmission of affinity values and the accumulation of evidence in data points. At the end, evidence is located only on a certain number of data points that are chosen as cluster centroids. Then, the set of classes is constructed by assigning each data point to its closest centroid. It should be noted that the centroids necessarily correspond to data points and that the number of classes does not need to be fixed in advance. That is, the number of classes will automatically emerge as the algorithm unfolds. The authors in [14] also show that AP approximately minimizes the sum of the squares of the Euclidean distance between each data point and its assigned centroid.

AP was applied using the centroids of the 200 classes produced by K-means as data points. At the end, these 200 classes were aggregated into 21 different classes, labeled from 0 to 20. Figures 5 and 6 show the number of arcs and observations, respectively, in each class.

It should be noted that it was not possible to apply AP directly to the $N = 3{,}485$ average speed patterns, which proved to be too large. This is the reason for the two-phase process, where K-means is applied first to reduce the problem size, followed by AP in the second phase. We also tested another clustering algorithm, known as the Mean Shift Algorithm [16], but since it proved to be worse than AP, we will omit its description.

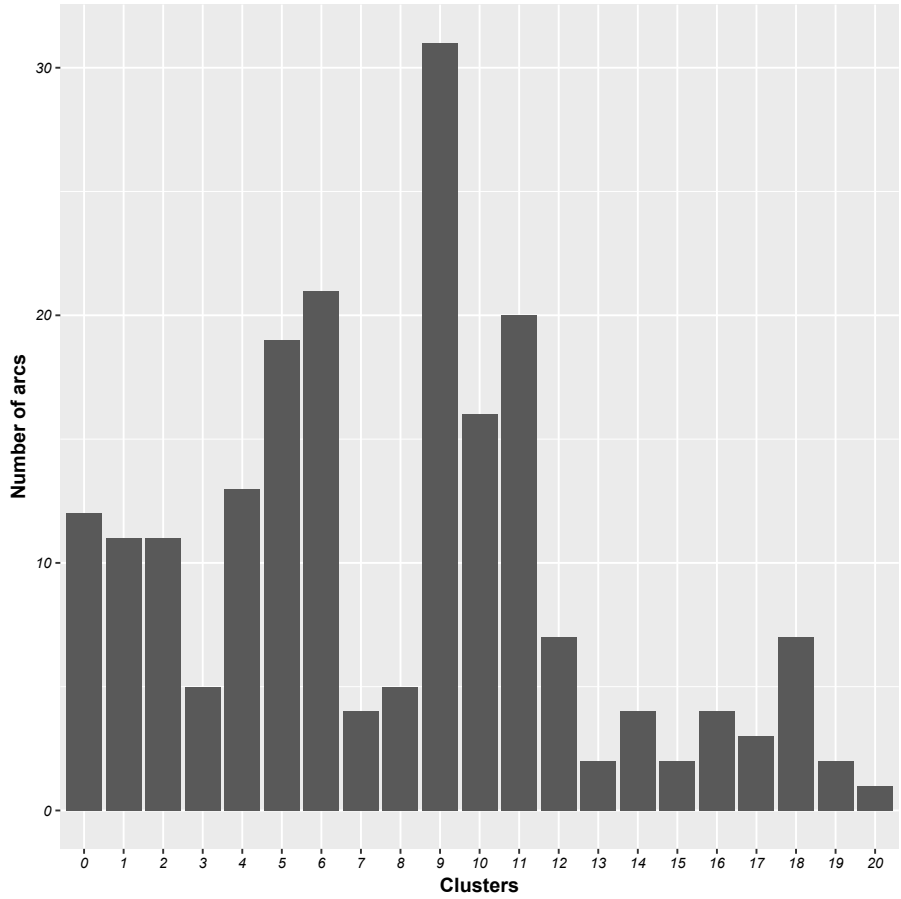

Figure 5 – Number of arcs in each class generated by AP

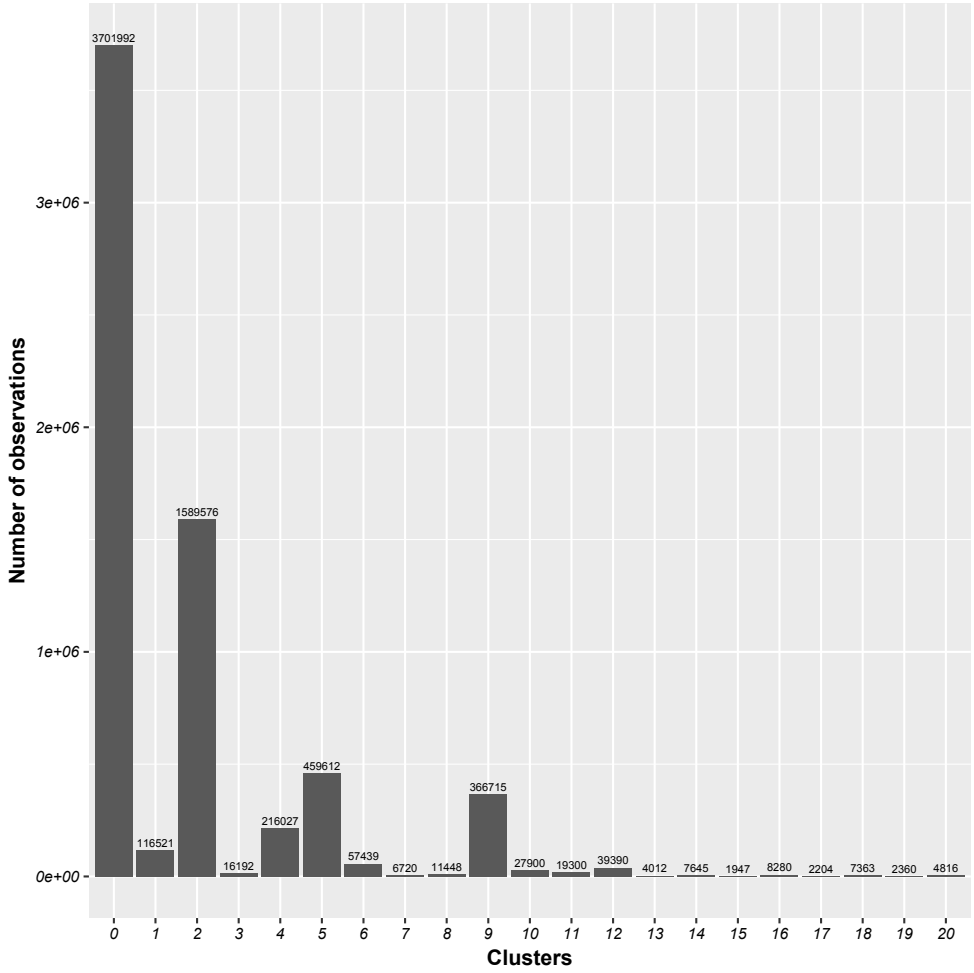

Figure 6 – Number of observations in each class generated by AP

4.3 Evaluation metrics

After clustering the arcs into 21 classes, the quality of these classes was evaluated using two well-known metrics, namely the Silhouette coefficient and the Calinski-Harabasz score.

The Silhouette coefficient [37] associates a value between -1 and 1 with each data point. It can be interpreted as follows: if the coefficient is close to 1, then the data point is associated with the right cluster; if it is close to 0, then it lies between two clusters; and if it is close to -1, then the point is not associated with the right cluster. Assuming that a data point corresponds to the average speed pattern associated with an arc, this coefficient is calculated as follows:

- for every arc i, calculate the average distance between its speed pattern and the speed pattern of all other arcs in the same class; we call this value a_i.
- for every arc i, calculate the average distance between its speed pattern and the speed patterns of all arcs in the closest cluster; we call this value b_i.
- the silhouette coefficient s_i of arc i is then:

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)} \ . \tag{7}$$

At the end, the final value is the average of those s_i coefficients over all arcs.

The Calinski-Harabasz score [7] is another measure that provides a ratio between intra-class and inter-classes dispersion values. The clusters are better defined when the score is higher. The score is computed as follows:

$$CH^{(K)} = \frac{Tr(B^{(K)})}{Tr(W^{(K)})} * \frac{N-K}{K-1} \ , \tag{8}$$

where

$$W^{(K)} = \sum_{k=1}^{K} \sum_{x \in C^{(k)}} (x - c^{(k)})(x - c^{(k)})^\intercal \ , \tag{9}$$

$$B^{(K)} = \sum_{k=1}^{K} n^{(k)} (c^{(k)} - c)(c^{(k)} - c)^\intercal \ . \tag{10}$$

In these equations, a vector should be viewed as a column. Also, K is the number of clusters or classes, N is the number of speed patterns (arcs), $W^{(K)}$ is the intra-cluster dispersion matrix, $B^{(K)}$ is the inter-cluster dispersion matrix, $Tr(M)$ is the trace of matrix M, $C^{(k)}$ is the set, of cardinality $n^{(k)}$, of speed patterns (arcs) in class k, $c^{(k)}$ is the centroid of speed patterns in class k and c is the centroid of all speed patterns.

Note that each entry (i, j) in matrices $W^{(K)}$ and $B^{(K)}$ corresponds to:

$$W_{ij}^{(K)} = \sum_{k=1}^{K} \sum_{x \in C^{(k)}} (x_i - c_i^{(k)})(x_j - c_j^{(k)}) \ , \tag{11}$$

$$B_{ij}^{(K)} = \sum_{k=1}^{K} n^{(k)} (c_i^{(k)} - c_i)(c_j^{(k)} - c_j) \ . \tag{12}$$

When the Silhouette coefficient was evaluated on the 21 classes produced by AP, a value of 0.85 was obtained. This is quite good, given that 1 is the best possible value (note that the coefficient for the clusters produced by the Mean Shift algorithm was equal to 0.67). Concerning the Calinski-Harabasz score, a value of 10.52 was obtained, which is to be compared with 3.39 for the Mean Shift algorithm.

By focusing on the four classes with the largest number of observations, namely classes 0, 2, 5 and 9, we observed that the average speeds of classes 0, 2 and 5 are very different, ranging from approximately 25 to 45 km/hour. The average speed of class 9 is similar to the one of class 2, but it does not evolve in the same way over the day. When we looked at more detailed data, we observed that the arcs of class 9 are not affected by the congestion observed during weekdays. That is, as opposed to classes 0, 2 and 5, there is no significant difference between the weekday average speeds and the weekend average speeds. Thus, the clustering algorithm was successful in identifying classes of arcs with different characteristics.

5 Speed prediction

This section describes the supervised neural network model for predicting travel speeds based on the classes generated by the AP clustering algorithm. Since a data missing issue emerges in this context, we will first explain how this problem is handled. Then, the neural network model will be described.

5.1 Missing data

Given that input vectors for the neural network model are obtained by averaging speeds over all arcs in a class produced by our clustering methodology, there is no missing data in the input (see Section 6.1). However, each target output vector corresponds to one of the 6,667,459 speed patterns in the database. Thus, it is likely for a target output to have one or more missing values.

To handle missing values, we must input plausible estimates drawn from an appropriate model. In this process, the following variables will be accounted for: day (Monday, Tuesday, ..., Saturday, Sunday), season (Spring, Summer, Fall, Winter), the arc's class label, and most importantly, the average speed in each time interval which corresponds to the variables with missing values. Different Multiple Imputation (MI) methods will be applied [38]. These methods generate multiple copies of an incomplete database and replace the missing values in each replicate with estimates drawn from some imputation method. An analysis is then performed on each complete database and a single MI estimate is calculated for each missing value by combining the estimates from the multiple complete databases [31, 38]. The meth-

ods considered here are: Multivariate Imputation via Chained Equation (MICE) [6], missForest (which relies on a Random Forest imputation algorithm [46]) and Amelia [25].

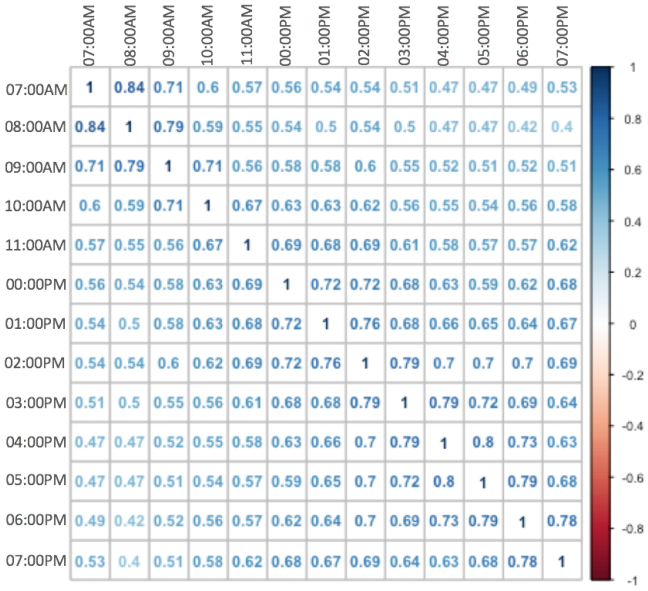

Figure 7 – Correlation matrix

5.1.1 MICE

This algorithm can be described as follows:

1. Perform a mean imputation for every missing speed value by setting it to the average over observed speeds in the same time interval.

2. Select the time interval variable with the largest proportion of missing speed values.

3. Select the explanatory variables from those with a correlation greater than 0.5 with the selected time interval variable (see, e.g., the correlation matrix in Figure 7).

4. Perform linear regression.

5. Replace the missing speed values for the selected time interval with estimates obtained from the regression model. If this time interval is subsequently used as

an explanatory variable in the regression model of other time interval variables, both observed and imputed values are used.

6. Repeat steps 2 to 4 for each remaining time interval with missing values (cycling through each variable stands for one iteration).

7. Repeat the entire procedure for a number of iterations to obtain multiple estimates.

At the end, the multiple estimates obtained over the iterations are averaged to obtain a single estimate for each missing value.

5.1.2 Random Forest

The Random Forest (RF) algorithm [5] is a machine learning technique that does not require the specification of a particular regression model. It has a built-in routine to handle missing values by weighting the observed values of a variable using a matrix of proximity values, where proximity is defined by the proportion of trees in which pairs of observations share a terminal node. It works as follows:

1. Replace missing values by the average over observed values in the same time interval.

2. Repeat until a stopping criterion is satisfied:

   (a) Using imputed values calculated so far, train a random forest.

   (b) Compute the proximity matrix.

   (c) Using proximity as a weight, impute missing values as the weighted average over observed values in the same time interval.

Typically, the algorithm stops when the difference between the new imputed values and the old ones increases for the first time. Note that, by averaging over multiple random trees, this method implicitly behaves according to a multiple imputation scheme. The RF algorithm was implemented using the randomForest R-package [30].

5.1.3 Amelia

Amelia imputes missing data based on different bootstrapped samples drawn from the database (bootstrapped data samples are large numbers of smaller samples of the same size that are repeatedly drawn, with replacement, from a single original sample, see [12]). It basically applies the Expectation Maximization (EM) method [11] to find the maximum likelihood estimates for the parameters of a distribution. Namely,

1. M samples are drawn from the original database.

2. M estimates of the mean and variance are calculated for each sample with EM.

3. Each estimate of the mean and variance is used to impute the missing data.

At the end, we have M different complete databases, where the missing values in each database have been filled with one of the M estimates of the mean and variance.

Each complete database produced by MICE, RF and Amelia is used to provide target outputs during the training and testing phases of our neural network model. The accuracy of the travel speed predictions made by the neural network will be compared for the three imputation methods considered.

5.2 LSTM

In this section, we briefly describe the supervised neural network model used to predict travel speeds, once the missing values in the database have been replaced by imputed ones (either using MICE, RF or Amelia). The neural network model is a Long Short-Term Memory network (LSTM). This choice was motivated by the ability of the LSTM to handle sequential data and to capture time dependencies. First introduced in [24], this special type of Recurrent Neural Network (RNN) alleviates the vanishing gradient problem [35] (when the gradient of the error function becomes too small with respect to a given weight, the latter cannot change anymore). It is made of an input layer, a variable number of hidden layers and an output layer. Each hidden layer is made of memory cells that store useful information from past input data. Memory cells in a given hidden layer send signals at time t to the memory cells in the next hidden layer (or the units in the output layer, if last) but also to themselves. This recurrent signal is used by the memory cells to determine their internal state at time $t + 1$. Thus, there are connection weight matrices from the input to the first hidden layer, from each hidden layer to itself and to the next hidden layer and from the last hidden layer to itself and to the output layer. Furthermore, memory cells in a given layer have three gates: one for the signal sent from the previous layer, one for the signal sent by the memory cells to themselves and one for the signal sent by the memory cells to the next layer. Gates can be seen as filters that regulate the signals by allowing some parts of it to be blocked (or forgotten). Like the weight matrices mentioned above, the gates have weights that are updated during the learning process. The interested reader will find more details about the LSTM network model in [19].

In the next section, computational results obtained with LSTM and comparisons with alternative approaches are reported.

6 Computational study

In this section, we first define the input and output vectors of the LSTM. Then, we describe the fine tuning of the LSTM hyperparameters. Finally, LSTM results are reported.

Our LSTM was implemented in Python 3.5. The hyperparameter tuning experiment was performed on a Dell R630 server with two Intel Xeon E5-2650V4 of 12 cores each (plus hyperthreading) and 256GB of memory. The server also has 4 NVIDIA Titan XP GPUs with 3840 CUDA cores and 12GB of memory. However, our code was limited to only 4 cores and one GPU from the server. To obtain more computation power, the final LSTM results reported in Sections 6.3 and 6.4 were obtained on the Cedar cluster of Compute Canada. We requested 6 cores with Intel E5-2650v4 processors, 32GB of RAM and 1 NVIDIA P100 GPU with 12GB of memory.

6.1 Input and output vectors

The input vector for the neural network was first designed as illustrated in Figure 8. Each vector is associated with a class of arcs produced by AP and is made of: the class label, the day (Monday, Tuesday, ..., Saturday, Sunday), the season (Spring, Summer, Fall, Winter) and 13 average speeds over all arcs in the corresponding class, that is, one speed value for each one-hour time interval starting from 7:00 AM to 7:00 PM. The target output vector corresponds to a speed pattern among the 6,667,459 available speed patterns, where the missing values are filled with one of the three imputation methods of Section 5.1. Obviously, the target output vector must come from an arc of the same class, and for the same season and day than the vector provided in input. We should also note that the speed values in the patterns were normalized using the scikit-learn object [36].

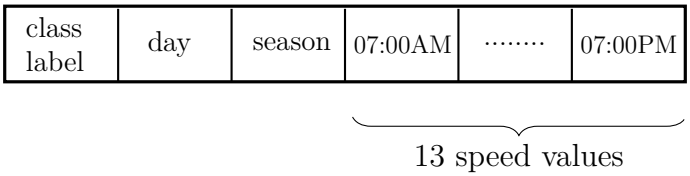

Figure 8 – Input vector I

Unfortunately, the results obtained with this approach were unsatisfactory. To better exploit the capabilities of LSTM to handle sequential data, we turned to input vector II shown in Figure 9. Here, input vector I with 13 speed values is transformed into 13 input vectors II, each with a single speed value. That is, rather than providing at once the whole speed pattern for one-hour time intervals starting from 7:00 AM

to 7:00 PM, a sequence of 13 input vectors from 7:00 AM to 7:00 PM is provided, where each input vector contains a single speed value. The target output vector is modified accordingly and also contains a single speed value taken from an arc of the same class, and for the same season, day and hour than the vector provided in input.

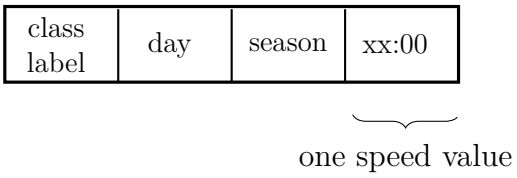

Figure 9 – Input vector II

6.2 Hyperparameter tuning

Our database of 6,667,459 speed patterns was divided into a training set (80% of the total) and a testing set (20% of the total), where the latter is made of the most recent observations. Apart from the connection weights, which are adjusted through learning, a neural network model also relies on a number of other parameters (where parameter is taken in a broad sense) that must be set before learning takes place. The following were considered:

- Number of hidden layers;
- Number of units in each hidden layer;
- Batch size: Number of training examples provided to the neural network before updating the connection weights;
- Training epochs: Number of passes through the set of training examples;
- Learning algorithm: Algorithm used during the training phase to adjust the weights;
- Weight initialization: Method used to set the initial weights, see [17] for more details;
- Activation function: Function used to compute the internal state of a unit from the signal it receives.

A good parameter setting for a neural network has a huge impact on its results, as discussed in [2]. To determine an appropriate combination of the above parameters, different hyperparameter optimization strategies can be used [45], in particular

grid search and random search. In grid search, a systematic evaluation of all possible combinations of parameter values is performed. Random search is much less computationally expensive, given that only a sample of all possible combinations is considered (100 combinations, in our case). Due to the curse of dimensionality, the superiority of random search over grid search is known for high-dimensional parameter spaces [3]. Since the number of parameters considered in our study is neither large nor small, both search methods were applied to our LSTM. The values tested for each parameter and the final setting obtained by each method are shown in Table 1. Note that the activation function differs depending on the hidden layer considered: the sigmoid function is used for the first hidden layer while the hyperbolic tangent (tanh) is used for the second and third hidden layers.

Table 1 – Hyperparameter tuning using grid search and random search

Hyperparameters	**Values**	**Grid search**	**Random search**
Hidden layers	1,2,3,4,5	3	3
Units in each hidden layer	1,5,10,15,20,25,30,35,40,45,50	20	20
Batch size	10,20,30,40,50,80,100	20	20
Training epochs	5,10,20,30,40,50	10	10
Learning algorithm	SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam	SGD	Adam
Activation function	softmax, softplus, softsign, relu, tanh, sigmoid, hard sigmoid, linear	sigmoid tanh tanh	sigmoid tanh tanh
Weight initialization	uniform, lecun uniform, normal, zero, glorot normal, glorot uniform, he normal, he uniform	normal	normal

Overall, the only difference between the two hyperparameter optimization methods is the learning algorithm. Grid search suggests Stochastic Gradient Descent (SGD) while random search suggests Adam. The latter, introduced by [29], is particularly appropriate for complex neural network structures and large datasets, as detailed in [39]. To get a better idea, we applied the various tested learning algorithms (see Table 1) with the other hyperparameters fixed at their best value on a subset of the database. The results obtained with the RMSE metric are shown in

Figure 10. Based on this preliminary experiment, we decided to go with Adam.

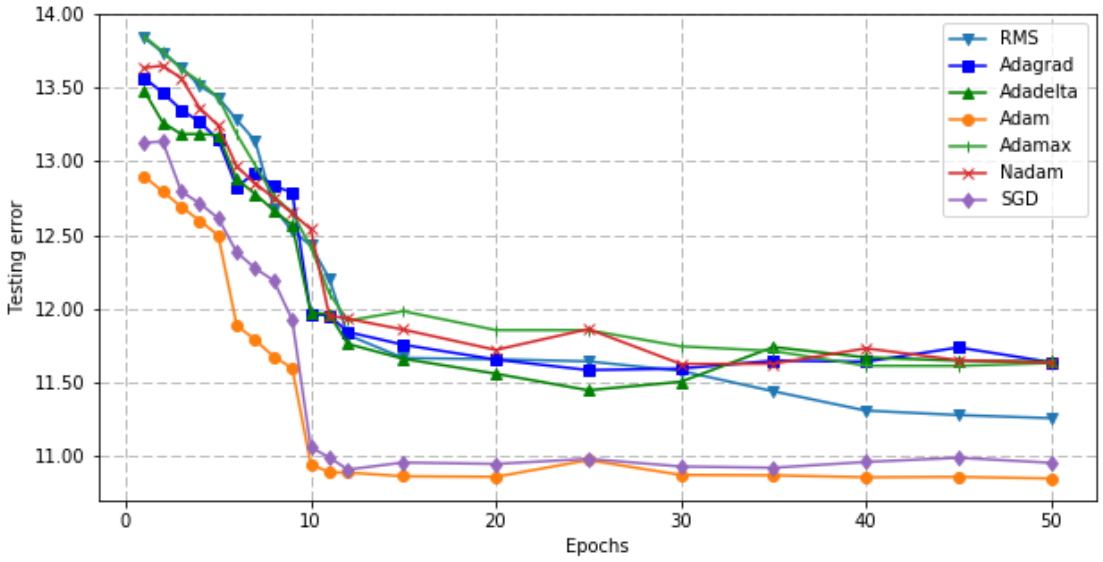

Figure 10 – Initial comparison of different learning algorithms

6.3 LSTM results

Here, we measure the accuracy of the travel speed predictions produced by our LSTM with three hidden layers. We also compare the results obtained with the three imputation methods of Section 5.1 to fill the missing values.

The root mean squared error and the mean absolute error are used to measure the accuracy, where:

$$RMSE = \sqrt{\frac{1}{L}\sum_{l=1}^{L}(y_l - \hat{y}_l)^2} \ , \tag{13}$$

$$MAE = \frac{1}{L}\sum_{l=1}^{L}|y_l - \hat{y}_l| \ . \tag{14}$$

In these equations, L is the number of (input, target output) pairs in the training or testing set, where the target output corresponds to an observed speed pattern. In pair l, y_l stands for the observed speed pattern and $\hat{y}_l$ for the speed pattern produced by the neural network for input vector l. RMSE and MAE are two error functions typically used to evaluate how close the output vectors produced by the

neural network are to the target outputs. Due to the square in the RMSE formula, large errors have much more impact than small ones. On the other hand, MAE measures the relative error and is more robust to outliers since there is no square in its formula.

Table 2 reports the prediction errors of the trained LSTM on the testing set, based on the RMSE and MAE metrics, using the three different imputation methods and a variable number of imputations (i.e., 5, 10, 15, 20). Note that the authors in [41] show that 3 to 5 imputations yield good results. But, more recently, the authors in [18, 40] proposed 20 imputations. Thus, we chose to vary this number between 5 and 20.

Table 2 – RMSE and MAE metrics with different imputation methods

Method	# Imput.	MAE	RMSE	Running time (min)
MICE	5	13.71	12.91	429
	10	**12.84**	**10.84**	450
	15	12.52	10.32	1117
	20	12.39	10.09	1900
RF	5	13.84	13.28	512
	10	13.02	11.38	649
	15	12.94	11.08	1640
	20	12.68	11.03	2000
AMELIA	5	17.91	17.53	550
	10	17.63	16.97	580
	15	16.76	16.10	940
	20	16.49	16.07	1500

The results show that MICE produces the best results. Although the two error metrics keep improving with the number of imputations, most of the improvement occurs between 5 and 10 imputations. Concerning the computing times, a substantial increase is observed between 10 and 15 imputations for all methods. Overall, MICE is the least computationally expensive for 5, 10 and 15 imputations. The marginal improvement obtained with 15 and 20 imputations probably does not justify the additional computational cost. Thus, MICE-10 seems to be a good compromise.

Figure 11 summarizes the evolution of the MAE and RMSE metrics for LSTM with MICE-10 over a number of training epochs. We can see that RMSE drops sooner than MAE and then keeps improving at about the same pace than MAE. Figure 12 then illustrates the differences between the observed speeds and the speeds predicted

by the trained LSTM on the training and testing (input, target output) pairs on a sample of observations. This figure shows that, in most cases, the predicted speed values follow closely the observed ones.

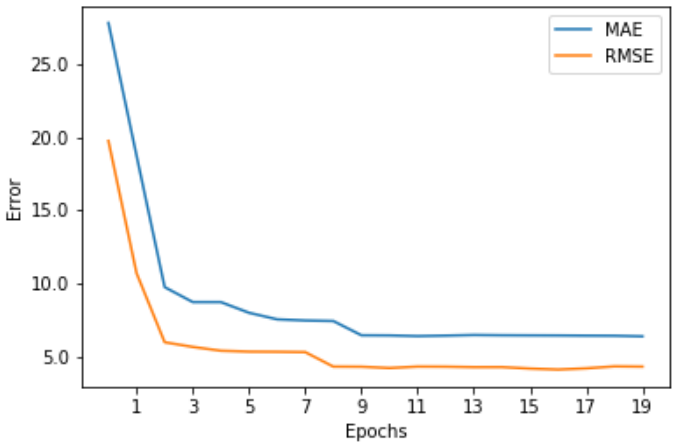

Figure 11 – Evolution of MAE and RMSE during the training phase

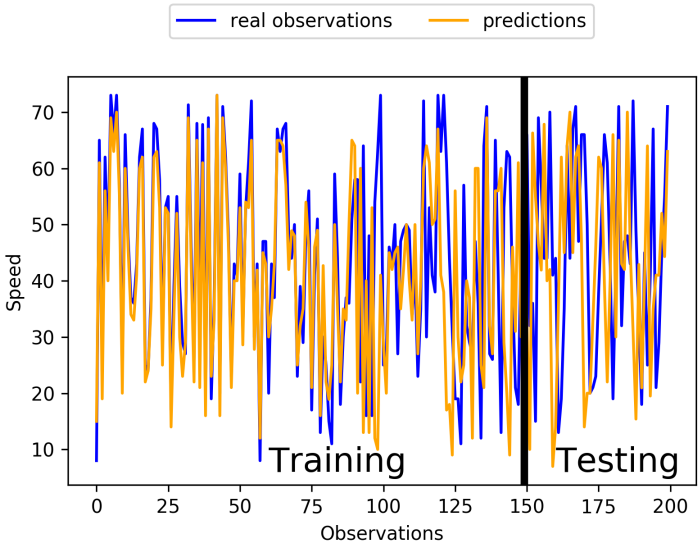

Figure 12 – Comparison between observed and predicted speed values

6.4 Comparison with other models

In this section, LSTM is compared with two alternative approaches based on a trivial average method and a Multi-Layer Perceptron (MLP) trained with the Levenberg-Marquardt algorithm [47]. We tested MLP models with one to three hidden layers, using a variable number of units in the hidden layers. At the end, the error metrics of the various models were quite close, although a model with three hidden layers and 30 units in each hidden layer proved to be the best. Thus, we only report the results for the best MLP model in Tables 3 and 4.

The average method is quite simple: if we consider an input vector of type I (II) for a given class, day and season (and hour), the output is the average speed pattern (value) over speed patterns (values) of arcs in the same class, for the same day and season (and hour). The results obtained with the LSTM, MLP and average method are reported in Tables 3 and 4 using the RMSE and MAE metrics, respectively. The results for the two input structures are also reported to show the superiority of input vector II over input vector I, see Section 6.1. Overall, LSTM clearly provides a better prediction accuracy than the two other models for both the RMSE and MAE metrics.

Table 3 – Comparison of alternative models: RMSE metric

Model	Input vector I	Input vector II
LSTM	**17.22**	**10.84**
MLP	21.92	15.17
Average	38.04	32.04

Table 4 – Comparison of alternative models: MAE metric

Model	Input vector I	Input vector II
LSTM	**22.86**	**12.84**
MLP	29.57	17.40
Average	42.62	36.98

7 Conclusion

In this work, a "prediction-after-classification" approach was proposed, starting from a large database of GPS traces collected from mobile devices installed inside delivery

vehicles. We used dimensionality reduction and unsupervised learning techniques to extract similarity classes of arcs to be used during the following supervised prediction phase.

Because supervised learning algorithms are sensitive to missing values, different multiple imputation methods were applied to obtain a complete database. The prediction problem was addressed with a LSTM neural network whose parameters were adjusted with random search. The LSTM was then compared to two alternative models and prove to be largely superior.

A natural extension of the work reported in this paper would be to include real-time data in the prediction process. The ultimate goal is to integrate the current predictor into a vehicle routing optimization procedure to produce more efficient delivery routes.

Acknowledgments. Financial support was provided by the Natural Sciences and Engineering Research Council of Canada through the Canada Excellence Research Chair in Data Science for Real-Time Decision-Making. Also, computing facilities were made available to us by Compute Canada. This support is gratefully acknowledged. Finally, we wish to thank Prof. Leandro C. Coelho and his team for their contribution to the analysis of the GPS data.

References

[1] C. Antoniou, M. Ben-Akiva, and H.N. Koutsopoulos. Nonlinear Kalman filtering algorithms for on-line calibration of dynamic traffic assignment models. *IEEE Transactions on Intelligent Transportation Systems*, 8(4):661–670, 2007.

[2] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

[3] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[4] George EP Box and Gwilym M Jenkins. *Time series analysis: forecasting and control, revised ed.* Holden-Day, 1976.

[5] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[6] S. Buuren and K. Groothuis-Oudshoorn. MICE: Multivariate imputation by chained equations in R. *Journal of Statistical Software*, 45(3), 2011.

[7] T. Calinski and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics*, 3(1):1–27, 1974.

[8] H. Chen, S. Grant-Muller, L. Mussone, and F. Montgomery. A study of hybrid neural network approaches and the effects of missing data on traffic forecasting. *Neural Computing & Applications*, 10(3):277–286, 2001.

[9] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[10] G.A. Davis and N.L. Nihan. Nonparametric regression and short-term freeway traffic forecasting. *Journal of Transportation Engineering*, 117(2):178–188, 1991.

[11] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B*, pages 1–38, 1977.

[12] B. Efron and R.J. Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[13] A. Ermagun and D. Levinson. Spatiotemporal traffic forecasting: Review and proposed directions. *Transport Reviews*, pages 1–29, 2018.

[14] B.J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[15] R. Fu, Z. Zhang, and L. Li. Using LSTM and GRU neural network methods for traffic flow prediction. In *Youth Academic Annual Conference of the Chinese Association of Automation*, pages 324–328. IEEE, 2016.

[16] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975.

[17] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[18] J.W. Graham, A.E. Olchowski, and T.D. Gilreath. How many imputations are really needed? Some practical clarifications of multiple imputation theory. *Prevention Science*, 8(3):206–213, 2007.

[19] K. Greff, R.K. Srivastava, J. Koutník, B.R. Steunebrink, and J. Schmidhhuber. LSTM: A search space odyssey. *IEEE Trans. on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017.

[20] J. Guo. Adaptive Estimation and Prediction of Univariate Vehicular Traffic Condition Series. *Ph.D. Thesis, North Carolina State University*, 2005.

[21] A.B. Habtie, A. Abraham, and D. Midekso. Artificial neural network based real-time urban road traffic state estimation framework. In *Computational Intelligence in Wireless Sensor Networks*, pages 73–97. Springer, 2017.

[22] M.M. Hamed, H.R. Al-Masaeid, and Z.M.B. Said. Short-term prediction of traffic volume in urban arterials. *Journal of Transportation Engineering*, 121(3):249–254, 1995.

[23] M. Hashemi and H.A. Karimi. A weight-based map-matching algorithm for vehicle navigation in complex urban networks. *Journal of Intelligent Transportation Systems*, 20(6):573–590, 2016.

[24] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[25] J. Honaker, G. King, and M. Blackwell. Amelia II: A program for missing data. *Journal of Statistical Software*, 45(7):1–47, 2011.

[26] H. Jula, M. Dessouky, and P.A. Ioannou. Real-time estimation of travel times along the arcs and arrival times at the nodes of dynamic stochastic networks. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):97–110, 2008.

[27] S.J. Julier and J.K. Uhlmann. New extension of the Kalman filter to nonlinear systems. In *AeroSense'97*, pages 182–193. International Society for Optics and Photonics, 1997.

[28] R.E. Kalman and R.S. Bucy. New results in linear filtering and prediction theory. *Journal of Basic Engineering*, 83(1):95–108, 1961.

[29] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] A. Liaw and M. Wiener. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[31] R.J.A. Little and D.B. Rubin. *Statistical Analysis with Missing Data*. John Wiley & Sons, 2014.

[32] X. Ma, Z. Tao, Y.i Wang, H. Yu, and Y. Wang. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, 54:187–197, 2015.

[33] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.

[34] M. Moniruzzaman, H. Maoh, and W. Anderson. Short-term prediction of border crossing time and traffic volume for commercial trucks: A case study for the Ambassador bridge. *Transportation Research Part C: Emerging Technologies*, 63:182–194, 2016.

[35] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[36] 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, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[37] P.J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[38] D.B. Rubin. *Multiple Imputation for Nonresponse in Surveys*. John Wiley & Sons, 2004.

[39] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[40] J.L. Schafer and J.W. Graham. Missing data: our view of the state of the art. *Psychological Methods*, 7(2):147, 2002.

[41] J.L. Schafer and M.K. Olsen. Multiple imputation for multivariate missing-data problems: A data analyst's perspective. *Multivariate Behavioral Research*, 33(4):545–571, 1998.

[42] B.L. Smith. *Forecasting Freeway Traffic Flow for Intelligent Transportation Systems Application*. PhD thesis, Charlottesville, VA, USA, 1995.

[43] B.L. Smith and M.J. Demetsky. Traffic flow forecasting: comparison of modeling approaches. *Journal of Transportation Engineering*, 123(4):261–266, 1997.

[44] B.L. Smith, B.M. Williams, and R.K. Oswald. Comparison of parametric and nonparametric models for traffic flow forecasting. *Transportation Research Part C: Emerging Technologies*, 10(4):303–321, 2002.

[45] J. Snoek, H. Larochelle, and R.P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

[46] D.J. Stekhoven and P. Bühlmann. Missforest-Non-parametric missing value imputation for mixed-type data. *Bioinformatics*, 28(1):112–118, 2011.

[47] I. Sutskever. Training Recurrent Neural Networks. *Ph D. Thesis, University of Toronto, Toronto, Canada*, 2013.

[48] Y. Tian and L. Pan. Predicting short-term traffic flow by long short-term memory recurrent neural network. In *IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, pages 153–158. IEEE, 2015.

[49] C.P. Van Hinsbergen, J.W. Van Lint, and F.M. Sanders. Short term traffic prediction models. In *Proceedings of the 14th World Congress on Intelligent Transportation Systems (ITS), Bejing, China*, 2007.

[50] J.W.C. Van Lint. Online learning solutions for freeway travel time prediction. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):38–47, 2008.

[51] V.N. Vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 10(5):988–999, 1999.

[52] V.N. Vladimir. *The Nature of Statistical Learning Theory*. Springer Heidelberg, 1995.

[53] E. Vlahogianni, M. G Karlaftis, and J.C. Golias. Optimized and meta-optimized neural networks for short-term traffic flow prediction: A genetic approach. *Transportation Research Part C: Emerging Technologies*, 13(3):211–234, 2005.

[54] J. Wang and Q. Shi. Short-term traffic speed forecasting hybrid model based on chaos–wavelet analysis-support vector machine theory. *Transportation Research Part C: Emerging Technologies*, 27:219–232, 2013.

[55] Y. Wang and M. Papageorgiou. Real-time freeway traffic state estimation based on extended Kalman filter: A general approach. *Transportation Research Part B: Methodological*, 39(2):141–167, 2005.

[56] B.M. Williams and L.A. Hoel. Modeling and forecasting vehicular traffic flow as a seasonal ARIMA process: Theoretical basis and empirical results. *Journal of Transportation Engineering*, 129(6):664–672, 2003.

[57] C.-H. Wu, J.-M. Ho, and D.-T. Lee. Travel-time prediction with support vector regression. *IEEE Transactions on Intelligent Transportation Systems*, 5(4):276–281, 2004.

[58] Z. Yang, D. Mei, Q. Yang, H. Zhou, and X. Li. Traffic flow prediction model for large-scale road network based on cloud computing. *Mathematical Problems in Engineering*, Volume 2014, 2014.