Big Data Research ••• (••••) •••–•••

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Train Delay Prediction Systems: A Big Data Analytics Perspective ☆

Luca Oneto [a,*], Emanuele Fumeo [a], Giorgio Clerico [a], Renzo Canepa [b], Federico Papa [c], Carlo Dambra [c], Nadia Mazzino [c], Davide Anguita [a]

[a] DIBRIS – University of Genova, Via Opera Pia 13, I-16145, Genova, Italy
[b] Rete Ferroviaria Italiana S.p.A., Via Don Vincenzo Minetti 6/5, I-16126, Genoa, Italy
[c] Ansaldo STS S.p.A., Via Paolo Mantovani 3-5, I-16151, Genova, Italy

ARTICLE INFO

Article history:
Received 9 January 2017
Received in revised form 20 April 2017
Accepted 14 May 2017
Available online xxxx

Keywords:
Railway network
Train Delay Prediction systems
Big data analytics
Extreme learning machines
Shallow architecture
Deep architecture

ABSTRACT

Current train delay prediction systems do not take advantage of state-of-the-art tools and techniques for handling and extracting useful and actionable information from the large amount of historical train movements data collected by the railway information systems. Instead, they rely on static rules built by experts of the railway infrastructure based on classical univariate statistic. The purpose of this paper is to build a data-driven Train Delay Prediction System (TDPS) for large-scale railway networks which exploits the most recent big data technologies, learning algorithms, and statistical tools. In particular, we propose a fast learning algorithm for Shallow and Deep Extreme Learning Machines that fully exploits the recent in-memory large-scale data processing technologies for predicting train delays. Proposal has been compared with the current state-of-the-art TDPSs. Results on real world data coming from the Italian railway network show that our proposal is able to improve over the current state-of-the-art TDPSs.

1. Introduction

Big Data Analytics is one of the current trending research interests in the context of railway transportation systems. Indeed, many aspects of the railway world can greatly benefit from new technologies and methodologies able to collect, store, process, analyze and visualize large amounts of data [1,2] as well as new methodologies coming from machine learning, artificial intelligence, and computational intelligence to analyze that data in order to extract actionable information [3]. Examples are: condition based maintenance of railway assets [4–6], automatic visual inspection systems [7,8], risk analysis [9], network capacity estimation [10], optimization for energy-efficient railway operations [11], marketing analysis for rail freight transportation [12], usage of ontologies and linked data in railways [13], big data for rail inspection systems [14], complex event processing over train data streams [15], fault diagnosis of vehicle on-board equipment for high speed railways [16–18] and for conventional ones [19], research on storage and retrieval of large amounts of data for high-speed trains [20], development of an online geospatial safety risk model for railway networks [21], train marshaling optimization through genetic algorithms [22], research on new technologies for the railway ticketing systems [23].

In particular, this paper focuses on building a Train Delay Prediction System (TDPS) in order to provide useful information to traffic management and dispatching processes through the usage of state-of-the-art tools and techniques, able to extract useful and actionable information from the large amount of historical train movements data collected by the railway information systems.

Delays can be due to various causes: disruptions in the operations flow, accidents, malfunctioning or damaged equipment, construction work, repair work, and severe weather conditions like snow and ice, floods, and landslides, to name just a few. Although trains should respect a fixed schedule called Nominal Timetable (NT), Train Delays (TDs) occur daily and can negatively affect railway operations, causing service disruptions and losses in the worst cases. Rail Traffic Management Systems (TMSs) [24] have been developed to support the management of the inherent complexity of rail services and networks by providing an integrated and holistic view of operational performance, enabling high levels of rail operations efficiency. By providing an accurate TDPS to TMSs, it is

☆ This article belongs to Big Data & Neural Network.

* Corresponding author.
E-mail addresses: luca.oneto@unige.it (L. Oneto), emanuele.fumeo@edu.unige.it (E. Fumeo), giorgio.clerico@edu.unige.it (G. Clerico), r.canepa@rfi.it (R. Canepa), federico.papa@ansaldo-sts.com (F. Papa), carlo.dambra.ext@ansaldo-sts.com (C. Dambra), nadia.mazzino@ansaldo-sts.com (N. Mazzino), davide.anguita@unige.it (D. Anguita).

http://dx.doi.org/10.1016/j.bdr.2017.05.002
2214-5796/

possible to greatly improve traffic management and dispatching in terms of:

- *Passenger information systems*, increasing the perception of the reliability of railway passenger services and, in case of service disruptions, providing valid alternatives to passengers looking for the best train connections [25].
- *Freight tracking systems*, estimating goods' time to arrival correctly in order to improve customers' decision-making processes.
- *NT planning*, providing the possibility of updating the train trip scheduling to cope with recurrent TDs [26].
- *Delay management (rescheduling)*, allowing traffic managers to reroute trains in order to utilize the railway network in a better way [27].

Due to its key role, a TMS stores the information about every Train Movement (TM), i.e. every train arrival and departure timestamp at "checkpoints" monitored by signaling systems (e.g. a station or a switch). Datasets composed by TM records have been used as fundamental data sources for every work addressing the problem of building a TDPS.

For instance, Milinkovic et al. [28] developed a Fuzzy Petri Net model to estimate TD based both on expert knowledge and on historical data. Berger et al. [29] presented a stochastic model for TD propagation and forecasts based on directed acyclic graphs. Pongnumkul et al. [30] worked on data-driven models for TD predictions, treating the problem as a time series forecast one. Their system was based on autoregressive integrated moving average and nearest neighbor models, although their work reports the application of their models over a limited set of data from a few trains. Finally, Kecman et al. [31–34] developed an intensive research in the context of TD prediction and propagation by using process mining techniques based on innovative timed event graphs, on historical TM data, and on expert knowledge about railway infrastructure.

However, their models are based on classical univariate statistics, while our solution integrates multivariate statistical concepts that allow our models to be extended in the future by including other kind of data (e.g. weather forecasts or passenger flows). Moreover, these models are not especially developed for Big Data technologies, possibly limiting their adoption for large scale networks.

For these reasons, this paper investigates the problem of predicting train delays for large scale railway networks by treating it as a time series forecast problem where every train movement represents an event in time, and by exploiting Big Data Analytics methodologies. Delay profiles for each train are used to build a set of data-driven models that, working together, make possible to perform a regression analysis on the past delay profiles and consequently to predict the future ones.

In the regression framework, and more in general in the supervised learning framework, Extreme Learning Machines (ELM) represent a state of the art tool [35]. ELM [36] were introduced to overcome problems posed by back-propagation training algorithm [36,37]: potentially slow convergence rates, critical tuning of optimization parameters, and presence of local minima that call for multi-start and re-training strategies. The original ELM are also called "Shallow" ELM (SELM) because they have been developed for the single-hidden-layer feedforward neural networks [36], and they have been generalized in order to cope with cases where ELM are not neuron alike. SELM were later improved to cope with problems intractable by shallow architectures [38] by proposing various Deep ELM (DELM) built upon a deep architecture [36,39], so to make possible to extract features by a multilayer feature representation framework. This work considers both SELM and DELM for predicting TDs, and proposes an adaptation of their typical learning

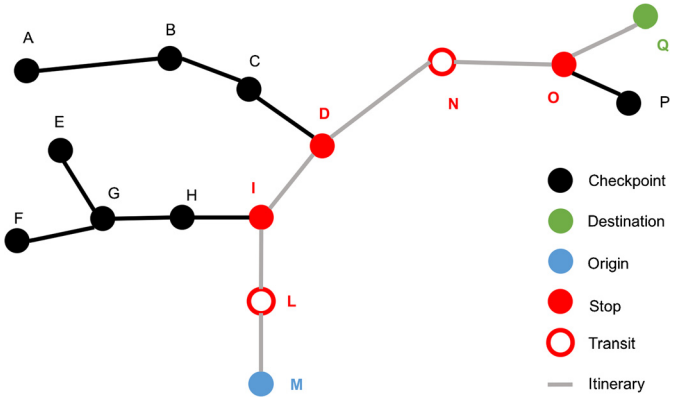

Fig. 1. A railway network depicted as a graph, including a train itinerary from checkpoint M to checkpoint Q.

strategies to exploit Big Data parallel architectures in order to meet the high-demanding requirements of Dynamic Large-Scale Railway Networks. In particular, the proposed implementations fully exploit the recent Apache Spark [40,41] in-memory large-scale data processing technology upon a state-of-art Big Data architecture [42] (Apache Spark on Apache YARN [43,44]) running on the Google Cloud infrastructure [45].

The described approach and the prediction system performance have been validated based on the real historical data provided by Rete Ferroviaria Italiana (RFI), the Italian Infrastructure Manager (IM) that controls all the traffic of the Italian railway network [46]. For this purpose, a set of novel Key Performance Indicators (KPIs) agreed with RFI and based on the requirements of their systems has been designed and used. Six months, from January 2016 to June 2016, of TM records from the entire Italian railway network have been exploited, showing that the new proposed methodology outperforms the current technique used by RFI, which is largely based on the state-of-the-art approach of [34], to predict TDs in terms of overall accuracy.

The paper is organized as follows. Section 2 presents the train delay prediction problem with particular reference to the Italian case. Section 3 describes the proposed train delay prediction systems based on shallow and deep Extreme Learning Machines. Section 4 describes the available data for building and testing the models based on a series of custom key index of performance developed with RFI. Section 5 reports the results and finally Section 6 concludes the paper.

2. Train delay prediction problem: the Italian case

A railway network can be considered as a graph where nodes represent a series of checkpoints consecutively connected. Any train that runs over the network follows an itinerary composed of n_c checkpoints $\mathcal{C} = \{C_1, C_2, \cdots, C_{n_c}\}$, which is characterized by a station of origin, a station of destination, some stops and some transits at checkpoints in between (see Fig. 1). For any checkpoint C, a train should arrive at time t_A^C and should depart at time t_D^C, defined in the NT. Usually time references included in the NT are approximated with a precision of 30 s or 1 min. The actual arrival and departure times of a train are defined as $\hat{t}_A^C$ and $\hat{t}_D^C$. The difference between the time references included in the NT and the actual times, either of arrival $(\hat{t}_A^C - t_A^C)$ or of departure $(\hat{t}_D^C - t_D^C)$, is defined as TD. Note that, in Italy, the RFI service contract states that if the TD of a train which runs on the RFI network is greater than 30 seconds or 1 minute (according to the commercial mission), then a train is considered as a delayed train. Note that, for the origin station there is no arrival time, while for the destination station there is no departure time. A dwell time is

Please cite this article in press as: L. Oneto et al., Train Delay Prediction Systems: A Big Data Analytics Perspective, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.05.002

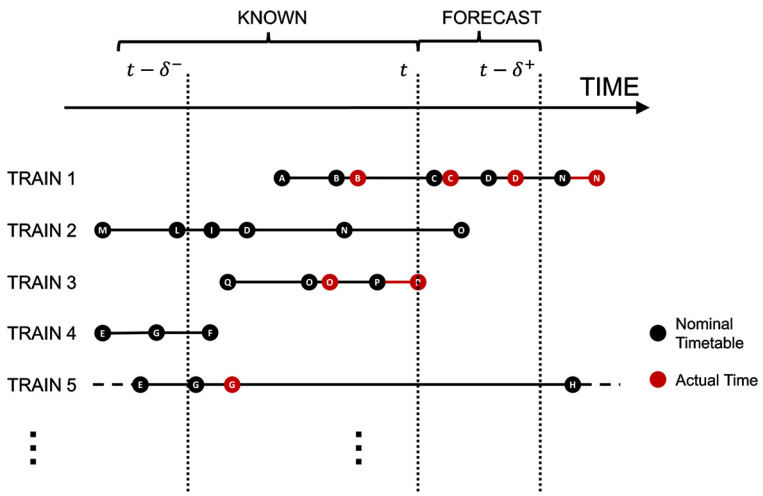

Fig. 2. Data available for the TD prediction models for the network of Fig. 1.

defined as the difference between the departure time and the arrival time for a fixed checkpoint ($\hat{t}_D^C - \hat{t}_A^C$), while a running time is defined as the amount of time needed to depart from the first of two subsequent checkpoints and to arrive to the second one ($\hat{t}_A^{C+1} - \hat{t}_D^C$).

In order to tackle the problem of building a TDPS, the following solution is proposed. Taking into account the itinerary of a train, the goal is to be able to predict the TD that will affect that specific train for each subsequent checkpoint with respect to the last one on which the train has transited. To make it general, for each checkpoint C_i, where $i \in \{0, 1, \cdots, n_c\}$, the prediction system must be able to predict the TD for each subsequent checkpoint $\{C_{i+1}, C_{i+2}, \cdots, C_{n_c}\}$. Note that C_0 is a virtual checkpoint that reproduces the condition of a train that still has to depart from its origin. In this solution, the TD prediction problem is treated as a time series forecast problem, where a set of predictive models perform a regression analysis over the TD profiles for each train, for each checkpoint C_i of the itineraries of these trains, and for each subsequent checkpoint C_j with $j \in \{i+1, i+2, \cdots, n_c\}$. Fig. 2 shows the data needed to build forecasting models based on the railway network depicted in Fig. 1. Basically, based on the state of the network between time $t - \delta^-$ and time t, the proposed system must be able to predict TD occurring from time t and $t + \delta^+$, and this is nothing but a classical regression problem.

Finally, it is worth noting that the intrinsic time varying nature of the delay phenomenon must be considered, which is due mainly to changes in the NT. This means that, in order to obtain good performances, the models should take into account only the amount of historical data representative of the actual distribution of the TD. For these reasons, considering a model built at day d_0 able to predict the TDs at day d_0+1, we have to rely on the historical data available between $d_0 - \Delta^-$ and d_0. Since Δ^- is a critical hyperparameter in the TD prediction problem, its value has been agreed with RFI experts, as it will be shown in Section 5. This choice was made based on many years of experience of the experts in RFI who deal with the problem of rescheduling the trains when a delay occurs. In particular, RFI experts observe that the delay behavior changes significantly only after a change in the nominal timetable (so Δ^- is set based on the last nominal timetable change).

To sum up, for each train characterized by a specific itinerary of n_c checkpoints, n_c models have to be built for C_0, $(n_c - 1)$ for C_1, and so on. Consequently, the total number of models to be built for each train can be calculated as $n_c + (n_c - 1) + \cdots + 1 = \frac{n_c(n_c-1)}{2}$. These models work together in order to make possible to estimate the TD of a particular train during its entire itinerary.

Considering the case of the Italian railway network, every day RFI controls approximately 10 thousand trains traveling along the national railway network. Every train is characterized by an itinerary composed of approximately 12 checkpoints, which means that the number of TMs is greater than or equal to 120 thousands per day. This results in roughly one message per second and more than 10 GB of messages per day to be stored. Note that every time that a complete set of TM records describing the entire planned itinerary of a particular train for one day is retrieved, the predictive models associated with that train must be retrained. The retraining phases can be performed at night, when only a few trains are traveling through the railway network and all the data of the just passed day is available, in order not to load the systems at daytime. Moreover, the continuous retraining of models allows both to cope with the intrinsic time dynamic nature of the system and to obtain the best possible performing model every new day. Since for each train at least $\frac{n_c(n_c-1)}{2} \approx 60$ models have to be built, the number of models that has to be retrained every day in the Italian case is greater than or equal to 600 thousands.

3. Train delay prediction systems

This section deals with the problem of building a data-driven TDPS. In particular, focusing on the prediction of the TD profile of a single train, there is a variable of interest (i.e. the TD profile of a train along with its itinerary) and other possible correlated variables (e.g. information about other trains traveling on the network or the day of the week). The goal is to predict the TD of that train at a particular time in the future $t = t + \delta^+$, i.e. at one of its following checkpoints. Due to the dynamic nature of the problem, only a part of the historical data have to be used (days in $[d_0 - \Delta^-, d_0]$), namely the most recent ones, which represent the distribution under exam. Given the previous observations, the TD prediction problem can be mapped into a classical time varying multivariate regression problem [47].

In the conventional regression framework [48] a set of data $\mathcal{D}_n = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$, with $\boldsymbol{x}_i \in \mathcal{X} \in \mathbb{R}^d$ and $y_i \in \mathcal{Y} \in \mathbb{R}$, is available from the automation system. The goal of the authors is to identify the unknown model $\mathfrak{S} : \mathcal{X} \to \mathcal{Y}$ through a model $\mathfrak{M} : \mathcal{X} \to \mathcal{Y}$ chosen by an algorithm $\mathscr{A}_{\mathcal{H}}$ defined by its set of hyperparameters $\mathcal{H}$. The accuracy of the model $\mathfrak{M}$ in representing the unknown system $\mathfrak{S}$ can be evaluated with reference to different measures of accuracy. In the case reported by this paper, they have been defined together with RFI experts, and have been reported in Section 4.

Please cite this article in press as: L. Oneto et al., Train Delay Prediction Systems: A Big Data Analytics Perspective, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.05.002

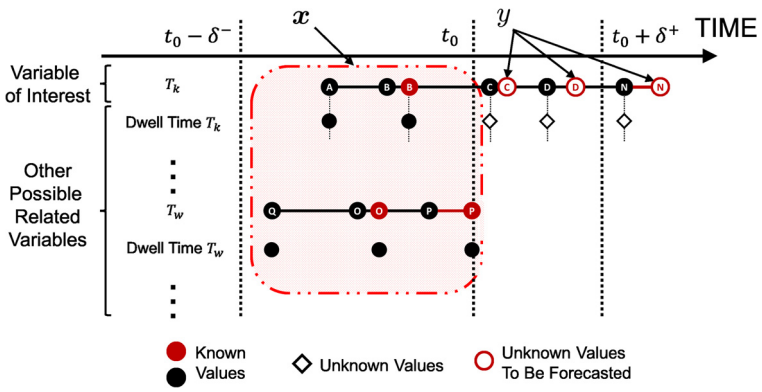

Fig. 3. Mapping of the TD prediction problem into a multivariate regression problem.

In order to map the TD prediction problem into a dynamic multivariate regression model, let us consider the train of interest T_k, which is at checkpoint $C_i^{T_k}$ with $i \in \{0, 1, \cdots, n_c\}$ at time t_0. The goal is to predict the TD at one of its subsequent checkpoints $C_j^{T_k}$, with $j \in \{i+1, i+2, \cdots, n_c\}$. Consequently, the input space $\mathcal{X}$ will be composed by:

- the current day of the week (Monday, Tuesday, ...);
- a boolean value indicating whether the current day is a holiday or a working day;
- the TDs, the dwell times and the running times for T_k for $t \in [t_0 - \delta^-, t_0]$;
- the TDs, the dwell times and the running times for all the other trains T_w with $w \neq k$ which were running over the same section of the railway network during the day for $t \in [t_0 - \delta^-, t_0]$.

Concerning the output space $\mathcal{Y}$, it is composed by $C_j^{T_k}$ with $j \in \{i+1, i+2, \cdots, n_c\}$ where $t_0 + \delta^+$ is equal to the NT of T_k for every $C_j^{T_k}$.

Fig. 3 shows a graphical representation of the mapping of the TD prediction problem into a multivariate regression problem. For instance, in this representation, the variable of interest is represented by the delay profile of T_k. The other possible related variables are represented by the information regarding all the other trains traveling along the network simultaneously to T_k. An example of $\boldsymbol{x} \in \mathcal{X}$ is highlighted in red. Analogously, all the aforementioned elements of the regression problem are depicted in the figure under examination.

Finally, $\mathcal{D}_n$ has to be built by exploiting the historical dataset composed of all the information collected during the days in $[d_0 - \Delta^-, d_0]$, so to cope with the dynamism of the problem.

3.1. Shallow extreme learning machines (SELM)

SELM were originally developed for the single-hidden-layer feedforward neural networks

$$f(\boldsymbol{x}) = \sum_{i=1}^{h} w_i g_i(\boldsymbol{x}), \tag{1}$$

where $g_i : \mathbb{R}^d \to \mathbb{R}$, $i \in \{1, \cdots, h\}$ is the hidden-layer output corresponding to the input sample $\boldsymbol{x} \in \mathbb{R}^d$, and $\boldsymbol{W} \in \mathbb{R}^h$ is the output weight vector between the hidden layer and the output layer.

In this case, the input layer has d neurons and connects to the hidden layer (having h neurons) through a set of weights $\boldsymbol{W} \in \mathbb{R}^{h \times (0, \cdots, d)}$ and a nonlinear activation function, $\varphi : \mathbb{R} \to \mathbb{R}$. Thus the i-th hidden neuron response to an input stimulus $\boldsymbol{x}$ is:

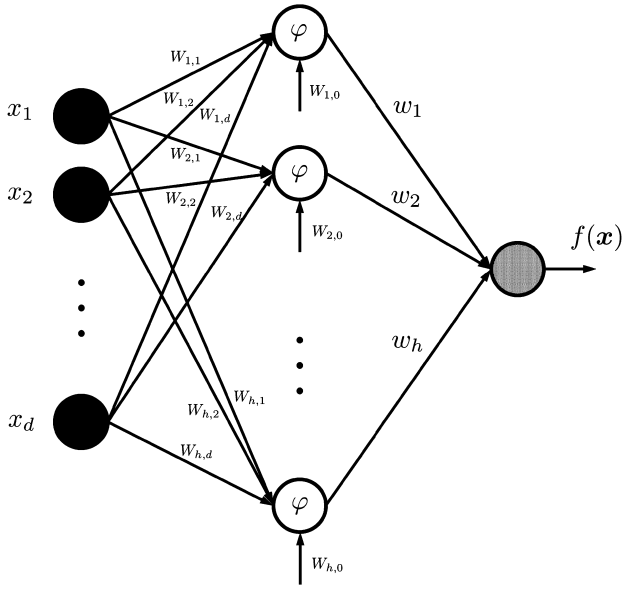

Fig. 4. SELM structure.

$$g_i(\boldsymbol{x}) = \varphi\left(\boldsymbol{W}_{i,0} + \sum_{j=1}^{d} \boldsymbol{W}_{i,j} x_j\right). \tag{2}$$

Note that Eq. (2) can be further generalized to include a wider class of functions [35]; therefore, the response of a hidden neuron to an input stimulus $\boldsymbol{x}$ can be generically represented by any nonlinear piecewise continuous function characterized by a set of parameters. In SELM, the parameters $\boldsymbol{W}$ are set randomly. A vector of weighted links, $\boldsymbol{W} \in \mathbb{R}^h$, connects the hidden neurons to the output neuron without any bias. The overall output function of the network (see Fig. 4) is:

$$f(\boldsymbol{x}) = \sum_{i=1}^{h} w_i \varphi\left(\boldsymbol{W}_{i,0} + \sum_{j=1}^{d} \boldsymbol{W}_{i,j} x_j\right) = \sum_{i=1}^{h} w_i \varphi_i(\boldsymbol{x}). \tag{3}$$

It is convenient to define an activation matrix, $\boldsymbol{A} \in \mathbb{R}^{n \times h}$, such that the entry $\boldsymbol{A}_{i,j}$ is the activation value of the j-th hidden neuron for the i-th input pattern. The $\boldsymbol{A}$ matrix is:

$$\boldsymbol{A} = \begin{bmatrix} \varphi_1(\boldsymbol{x}_1) & \cdots & \varphi_h(\boldsymbol{x}_1) \\ \vdots & \ddots & \vdots \\ \varphi_1(\boldsymbol{x}_n) & \cdots & \varphi_h(\boldsymbol{x}_n) \end{bmatrix}. \tag{4}$$

Please cite this article in press as: L. Oneto et al., Train Delay Prediction Systems: A Big Data Analytics Perspective, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.05.002

In the SELM model the weights $\boldsymbol{W}$ are set randomly and are not subject to any adjustment, and the quantity $\boldsymbol{W}$ in Eq. (3) is the only degree of freedom. Hence, the training problem reduces to minimization of the convex cost:

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} \|\boldsymbol{A}\boldsymbol{W} - \boldsymbol{y}\|^2 . \tag{5}$$

A matrix pseudo-inversion yields the unique L_2 solution:

$$\boldsymbol{W}^* = \boldsymbol{A}^+ \boldsymbol{y}. \tag{6}$$

The simple, efficient procedure to train a SELM therefore involves the following steps: (I) Randomly generate hidden node parameters (in this case $\boldsymbol{W}$); (II) Compute the activation matrix $\boldsymbol{A}$ (Eq. (4)); (III) Compute the output weights (Eq. (6)).

Despite the apparent simplicity of the SELM approach, the crucial result is that even random weights in the hidden layer endow a network with notable representation ability. Moreover, the theory derived in [36] proves that regularization strategies can further improve the approach's generalization performance. As a result, the cost function of Eq. (5) is augmented by a regularization factor [36]. A common approach is then to use the L_2 regularizer

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} \|\boldsymbol{A}\boldsymbol{W} - \boldsymbol{y}\|^2 + \lambda \|\boldsymbol{W}\|^2 , \tag{7}$$

and consequently the vector of weights $\boldsymbol{W}^*$ is then obtained as follows:

$$\boldsymbol{W}^* = (\boldsymbol{A}^T\boldsymbol{A} + \lambda I)^{-1}\boldsymbol{A}^T\boldsymbol{y}, \tag{8}$$

where $I \in \mathbb{R}^{h\times h}$ is an identity matrix. Note that h, the number of hidden neurons, is a hyperparameter that needs to be tuned based on the problem under exam.

Based on these considerations, it is possible to detect two main problems that would limit the application of SELM for building a TDPS:

- the first issue is that finding the solution of Eq. (7) through the approach of Eq. (8) is not efficient if n or h are large;
- the second issue is that, based on the description reported in Section 3, we have to explore all the historical information about the TMs looking for the right portion of data in order to build D_n. Scanning all the data for extracting the right one depending on T_k, δ^+, δ^-, and Δ^- must be as most efficient as possible.

These two issues can be solved by adopting a parallel optimization method for the solution of Eq. (7) and a state of the art technology for storing and processing large amount of data.

The first issue can be solved, by resorting to a Stochastic Gradient Descent (SGD) algorithm. The SGD algorithm is a very general optimization algorithm, which is able to solve a problem in the form of Eq. (8) efficiently [41]. Algorithm 1 reports the SGD algorithm for solving Eq. (7), where τ and n_{iter} are parameters related with the speed of the optimization algorithms. Therefore, usually τ and n_{iter} are set based on the experience of the user. In any case τ and n_{iter} can be seen as other regularization terms as λ since they are connected with the early stopping regularization technique [49].

Note that Algorithm 1 it suitable for being implemented with the Apache Spark technology [41]. Apache Spark is designed to efficiently deal with iterative computational procedures that recursively perform operations over the same data, such as in Algorithm 1. Moreover, one of the main ideas behind the Apache Spark technology [41,40] is to reduce the accesses to the disk as much as possible and instead to operate in memory. For this reason, Apache Spark is also useful for solving the second issue related to the application of SELM for building a DTDPS. Indeed, Spark allows to dramatically reduce the large number of disk accesses (necessary to build $\mathcal{D}_n$) by keeping, based on the available volatile memory, as much data as possible in memory, consequently speeding up the creation of different datasets $\mathcal{D}_n$ for different values of T_k, δ^+, δ^-, and Δ^-.

Algorithm 1: SGD for SELM.

Input: $\mathcal{D}_n$, λ, τ, n_{iter}
Output: $\boldsymbol{W}$
Read $\mathcal{D}_n$;
Compute $\boldsymbol{A}$;
$\boldsymbol{W} = \boldsymbol{0}$;
for $t \leftarrow 1$ **to** n_{iter} **do**
  $\boldsymbol{W} = \boldsymbol{W} - \frac{\tau}{\sqrt{t}} \frac{\partial}{\partial \boldsymbol{W}} \left[\|\boldsymbol{A}\boldsymbol{W} - \boldsymbol{y}\|^2 + \lambda \|\boldsymbol{W}\|^2\right]$;
return $(\boldsymbol{W}, b)$;

Algorithm 1 is well-suited for implementation in Spark and many of these tools are already available in MLlib [41]. Basically, the implementation of Algorithm 1 reported in Algorithm 2 is an application of two functions: a map for the computation of the gradient and a reduction function for the sum of each single gradient.

Algorithm 2: SGD for SELM on Spark ($d \geq h$).

Input: $\mathcal{D}_n$, λ, τ, n_{iter}
Output: $\boldsymbol{W}$
Read $\mathcal{D}_n$;
Compute $\boldsymbol{A}$ `/* Compute the projection` $\boldsymbol{\phi}$ `*/`
$\boldsymbol{W} = 0$;
for $t \leftarrow 1$ **to** n_{iter} **do**
  $\boldsymbol{g} = (\boldsymbol{A}, \boldsymbol{y})$.map(Gradient())
  `/* Compute the gradient for each sample */`
  .reduce(Sum())
  `/* Sum all the gradients of each sample */`
  $\boldsymbol{W} = \boldsymbol{W} - \frac{\tau}{\sqrt{t}} \boldsymbol{g}$;
return $\boldsymbol{W}$;

The main problem of Algorithm 2 is the computation and storage of $\boldsymbol{A}$. If $h \ll d$ it means that $\boldsymbol{A} \in \mathbb{R}^{n\times h}$ will be much smaller than the dataset which belongs to $\mathbb{R}^{n\times d}$. In this case, it is more appropriate to compute it before the SGD algorithms starts the iterative process and keep it in memory (note that the computation of $\boldsymbol{A}$ is fully parallel). In this way all the data $\mathbb{R}^{n\times d}$ projected by $\boldsymbol{\phi}$ into to matrix $\boldsymbol{A} \in \mathbb{R}^{n\times h}$ can be largely kept in volatile memory (RAM) instead of reading from the disk. If instead $h \gg d$, employing Algorithm 2 we risk that $\boldsymbol{A} \in \mathbb{R}^{n\times h}$ does not fit into the RAM, consequently making too many accesses to the disk. For this reason, we adopt two different strategies:

- if h is approximately the same magnitude or smaller than d, we use Algorithm 2 and we compute the matrix $\boldsymbol{A}$ at the beginning;
- if $h \gg d$, we adopt Algorithm 3 where $\boldsymbol{\phi}(\boldsymbol{x}_i)$ is computed online in order to avoid to read the data from the disk.

Quite obviously, the limit is given by the size of the RAM of each node and the number of nodes. Until the algorithm is able to keep most of the data in memory, it is better to use Algorithm 2. Algorithm 3 allows us to partially reduce the effect of having to access the data on the disk by paying the price of computing $\boldsymbol{\phi}(\boldsymbol{x}_i)$ online. In fact, Algorithm 3 does not precompute $\boldsymbol{A} \in \mathbb{R}^{n\times h}$ at the beginning but it keeps the data $\mathcal{D}_n$ in memory and, at every iteration of the SGD algorithm, it computes online both the projection induced by $\boldsymbol{\phi}$ and the gradient. Consequently, there is no need to store $\boldsymbol{A} \in \mathbb{R}^{n\times h}$.

Please cite this article in press as: L. Oneto et al., Train Delay Prediction Systems: A Big Data Analytics Perspective, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.05.002

Algorithm 3: SGD for SELM on Spark ($d \leq h$).

Input: $\mathcal{D}_n$, λ, τ, n_{iter}
Output: $\boldsymbol{W}$
Read $\mathcal{D}_n$;
$\boldsymbol{W} = 0$;
for $t \leftarrow 1$ **to** n_{iter} **do**
  $\boldsymbol{g} = \mathcal{D}_n$.map($\boldsymbol{\phi}$&Gradient())
  /* Compute both the projection $\boldsymbol{\phi}$ and the gradient for each sample */
  .reduce(Sum())
  /* Sum all the gradients of each sample */
  $\boldsymbol{W} = \boldsymbol{W} - \frac{\tau}{\sqrt{t}}\boldsymbol{g}$;
return $\boldsymbol{W}$;

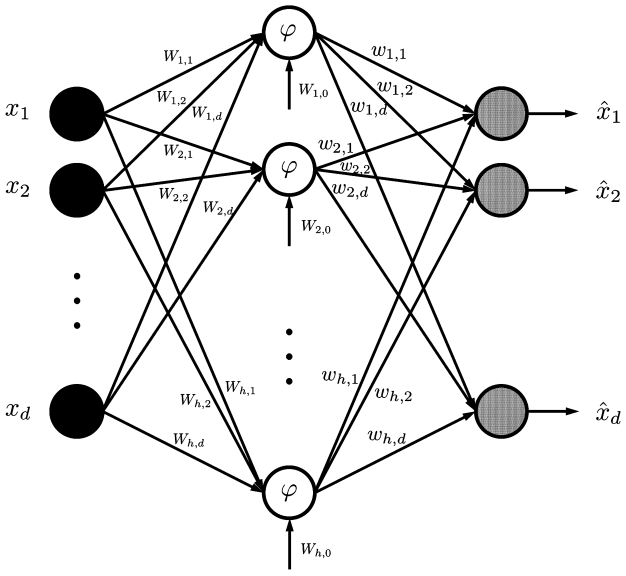

Fig. 5. DELM AE block.

3.2. Deep extreme learning machines (DELM)

Due to its shallow architecture, feature learning using SELM may not be effective even with a large number of hidden nodes. Since feature learning is often useful to improve the accuracy of the final model, multilayer (deep) solutions are usually needed. In [36] a multilayer learning architecture is developed using ELM-based autoencoder (AE) as its building block, which results in a sort of "Deep" ELM (DELM). The original inputs are decomposed into multiple hidden layers l, each one composed of $h_{i\in\{1,\cdots,l\}}$ hidden neurons, and the outputs of the previous layer are used as the inputs of the current one (see Fig. 5). Basically, instead of having just one output, we have a series of outputs $\hat{x}_j$ with $j \in \{1,\cdots,d\}$ such that

$$\hat{x}_j = f_j(\boldsymbol{x}) = \sum_{i=1}^{h} w_{j,i}\varphi\left(\boldsymbol{W}_{i,0} + \sum_{j=1}^{d}\boldsymbol{W}_{i,j}x_j\right) = \sum_{i=1}^{h} w_{j,i}\varphi_i(\boldsymbol{x}), \tag{9}$$

where $w_{j,i}$ with $j \in \{1,\cdots,d\}$ are found with the same approach of SELM. Before the supervised regularized least mean square optimization, the encoded outputs are directly fed into the last layer for decision making, without random feature mapping.

However, the approach developed in [36] does not fully exploit the potential of a multilayer implementation of ELM. Indeed, a new more powerful architecture that exploits the potential of a DELM is presented in [39], which considers multilayer as a whole with unsupervised initialization like in the classical Deep Learning approaches. After the unsupervised initialization, the entire network is trained by back propagation, and all the layers are hard coded together [38]. Note that, as for SELM, DELM do not require fine-tuning for the entire system, and consequently the training speed can be faster than the traditional back propagation based Deep Learning. Nevertheless, when big data problems are faced, using distributed computing and multiple GPUs can improve the speed of the back propagation based training phase of conventional Deep Neural Networks [50].

Although the approach of [39] can be much more effective than the one of [36], it requires more complex and time consuming computations. Instead, the approach described in [36] can produce improved results over the simple SELM since the number of hidden layers increases, and its implementation for big data problems can directly exploit the results of the previous section. Consequently, with reasonably small modifications, we are able to take advantage of a simple deep architecture by exploiting only the tools presented in Section 3.1.

3.3. Model selection

MS deals with the problem of tuning the performance of a learning procedure by tuning the hyperparameters of any learning algorithm [51]. Resampling techniques like hold out, cross validation and bootstrap [51] are often used by practitioners because they work well in many situations. Nevertheless other methods exist in literature. For example, [48] is the seminal work on Vapnik–Chervonenkis Dimension, which states the conditions under which a set of hypothesis is learnable. Later these results have been improved with the introduction of the Rademacher Complexity [51]. The theory of [52], was another step forward in the direction of understanding the learning properties of an algorithm by tightly connecting compression to learning. A breakthrough was made with the Algorithmic Stability [53], which states the properties that a learning algorithm should fulfill in order to achieve good generalization performance. The PAC-Bayes theory represents another fundamental brick [54] for MS, especially in the context of ensemble methods. Indeed, although it is well known that combining the outputs of a set of different learning procedures gives much better results than considering those learning procedures separately, it is hard to combine them appropriately in order to obtain satisfactory performances and it is not trivial to assess the performance of the resulting learning procedure. Finally, Differential Privacy (DP) allowed to reach a milestone result by connecting the privacy preservation in data analysis and the generalization capability of a learning algorithm. From one hand, it proved that a learning algorithm which shows DP properties also generalizes [55]. From the other hand, if an algorithm does not hold DP, it allows to state the conditions under which a hold out set can be reused without risk of false discovery through a DP procedure called Thresholdout [55].

In this paper we will use the 10-Fold Cross Validation [56,51] method in order to tune the hyperparameters of SELM and DELM. In particular, for SELM we have that $h \in \{1,2,\cdots\}$, $\lambda \in [0,\infty)$, and $\Delta^- \in \{1,2,\cdots\}$ days must be tuned, while for DELM we have to find the optimal values of $l \in \{1,2,\cdots\}$, $h_{i\in\{1,\cdots,l\}} \in \{1,2,\cdots\}$, $\lambda \in [0,\infty)$, and $\Delta^- \in \{1,2,\cdots\}$ days. Since it is not possible to fully explore all the combinations of hyperparameters, a search for the best set of hyperparameters over a finite grid of points is performed. Since we are dealing with a large amount of data and a large number of models to train, this approach results computationally intractable. Consequently, the approach of [57] has been selected, which consists in performing a random search by trying n_{MC} combinations of the hyperparameters. In [57] it is also shown that, both empirically and theoretically, randomly chosen trials are more efficient than trials on a grid.

4. Description of data and custom KPIs

In order to validate the proposed methodology and to assess the performance of the new prediction system, a large number of experiments have been performed on the real data provided by RFI. The Italian IM owns records of the TM from the entire Italian railway network over several years. For the purpose of this work, RFI gave access to six months of data related to the entire Italian railway network.

Each record refers to a single TM, and is composed by the following information: Date, Train ID, Checkpoint ID, Checkpoint Name, Arrival Time, Arrival Delay, Departure Time, Departure Delay and Event Type. The last field, namely "Event Type", refers to the type of event that has been recorded with respect to the train itinerary. For instance, this field can assume four different values: Origin (O), Destination (D), Stop (F) and Transit (T). The Arrival (Departure) Time field reports the actual time of arrival (departure) of a train at a particular checkpoint. Combining this information with the value contained in the Arrival (Departure) Delay field, it is possible to retrieve the scheduled time of arrival (departure). Note that, although IMs usually own proprietary software solutions, this kind of data can be retrieved by any rail TMS, since systems of this kind store the same raw information but in different formats. For example, some systems provide the theoretical time and the TD of a train, while others provide the theoretical time and the actual time, making the two information sets exchangeable without any loss of information. Finally, note that the information has been anonymized for privacy and security concerns.

The approach used to perform the experiments consisted in (i) building the needed set of models based on SELM and DELM for each train in the dataset, (ii) simultaneously tuning the models' hyperparameters through suitable models selection methodologies, (iii) applying the models to the current state of the trains, and finally (iv) validating the models in terms of performance based on what had really happened at a future instant. Consequently, simulations have been performed for all the trains included in the dataset adopting an online-approach that updates predictive models every day, in order to take advantage of new information as soon as it becomes available.

The results of the simulations have been compared with the results of the current TD prediction system used by RFI. The RFI system is quite similar to the one described in [34], although the latter includes process mining refinements which potentially increase its performance.

In order to fairly assess the performance of the proposed prediction system, a set of novel KPIs agreed with RFI has been designed and used. Since the purpose of this work was to build predictive models able to forecast the TD, these KPIs represent different indicators of the quality of these predictive models. Note that the predictive models should be able to predict, for each train and at each checkpoint of its itinerary, the TD that the train will have in any of the successive checkpoints. Based on this consideration, three different indicators of the quality of predictive models have been used, which are also proposed in Fig. 6 in a graphical fashion:

- *Average Accuracy at the i-th following Checkpoint for train j (AAiCj):* for a particular train j, the absolute value of the difference between the predicted delay and its actual delay is averaged, at the i-th following Checkpoint with respect to the actual Checkpoint.
- *AAiC:* is the average over the different trains j of AAiCj.
- *Average Accuracy at Checkpoint-i for train j (AACij):* for a particular train j, the average of the absolute value of the difference between the predicted delay and its actual delay, at the i-th checkpoint, is computed.

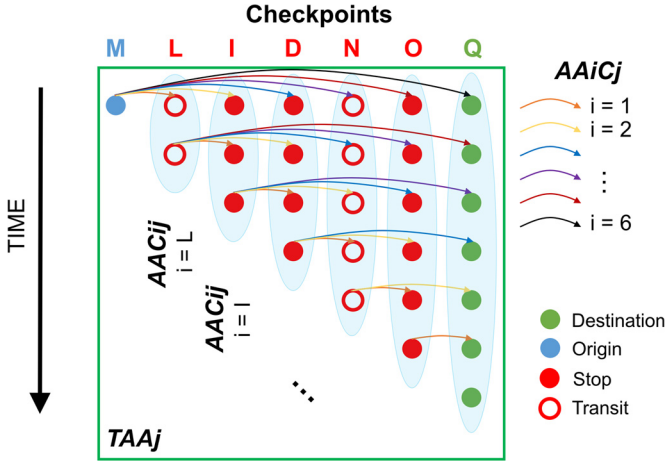

Fig. 6. KPIs for the train and the itinerary of Fig. 1.

- *AACi:* is the average over the different trains j of AACij.
- *Total Average Accuracy for train j (TAAj):* is the average over the different checkpoints i-th of AASij (or equivalently the average over the index i of AAiSj).
- *TAA:* is the average over the different trains j of TAAj.

5. Results

This section reports the results of the experiments exploiting the approaches described in Section 3, benchmarked with the data and KPIs described in Section 4.

The performance of different methods for building a TDPS have been compared:

- RFI: the RFI system has been implemented, which is quite similar to the one described in [34]. Note that, the RFI method neither exploits weather information nor has hyperparameters to tune;
- SL: SELM has been exploited (see Section 3.1), where the set of possible configurations of hyperparameters is searched in $h \in \{1, 2, \cdots, 10^4\}$ and $\lambda \in [10^{-}6, 10^4]$;
- DL: DELM has been exploited (see Section 3.2), where the set of possible configurations of hyperparameters has been defined as $l \in \{1, 2, \cdots, 10\}$, $h_{i \in \{1, \cdots, l\}} \in \{1, 2, \cdots, 10^4\}$, and $\lambda \in [10^{-}6, 10^4]$;

Note that $d_0 - \Delta^-$ is set equal to the time of the last change in the NT, and $t_0 - \delta^-$ is set equal to the time, in the NT, of the origin of the train, as suggested by the RFI experts.

Finally, as described in Section 3.3, 10 Fold Cross Validation technique has been exploited in order to optimize the different hyperparameters of the learning algorithms. The random search in the space of the hyperparameters has been done by setting $n_{MC} = 300$.

Tables 1, 2 and 3 report the KPIs of the different methods in the different scenarios. Note that the Tables are not complete due to space constraints, and that the train and station IDs have been anonymized because of privacy issues. In particular, it is possible to draw up the following comments:

- Table 1 reports the AAiCj and AAiC. From Table 1 it is possible to observe that the DELM method is the best performing method, and it improves up to $\times 2$ the current RFI system. All the data-driven methods (both SELM and DELM) improve over the RFI system by a large amount. The effects of this difference on the operations can be noticed not only by the travelers, but

Please cite this article in press as: L. Oneto et al., Train Delay Prediction Systems: A Big Data Analytics Perspective, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.05.002

Please cite this article in press as: L. Oneto et al., Train Delay Prediction Systems: A Big Data Analytics Perspective, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.05.002

Table 1
ELM based and RFI prediction systems KPIs (in minutes).

j	i														
	1st			2nd			3rd			4th			5th		
AAiCj	RFI	SELM	DELM	RFI	SELM	DELM	RFI	SELM	DELM	RFI	SELM	DELM	RFI	SELM	DELM
1	1.8 ± 0.5	1.6 ± 0.1	$\mathbf{1.5 \pm 0.2}$	2.1 ± 0.2	1.8 ± 1.3	$\mathbf{1.7 \pm 0.3}$	2.3 ± 0.5	2.1 ± 0.2	$\mathbf{1.9 \pm 0.6}$	2.5 ± 1.5	2.3 ± 1.3	$\mathbf{2.1 \pm 0.6}$	2.7 ± 0.0	2.4 ± 1.1	$\mathbf{2.3 \pm 0.3}$
2	3.2 ± 0.9	1.8 ± 0.6	$\mathbf{1.7 \pm 0.4}$	3.4 ± 0.7	$\mathbf{1.9 \pm 0.9}$	$\mathbf{1.9 \pm 0.7}$	3.8 ± 0.2	2.2 ± 0.3	$\mathbf{2.1 \pm 1.4}$	4.2 ± 1.8	2.4 ± 0.3	$\mathbf{2.3 \pm 0.3}$	4.6 ± 2.0	2.6 ± 0.9	$\mathbf{2.5 \pm 1.1}$
3	1.9 ± 0.2	1.4 ± 1.3	$\mathbf{1.3 \pm 0.4}$	2.0 ± 0.7	1.6 ± 0.1	$\mathbf{1.4 \pm 0.5}$	2.3 ± 0.7	1.8 ± 0.2	$\mathbf{1.7 \pm 0.3}$	2.6 ± 1.0	1.9 ± 0.2	$\mathbf{1.8 \pm 0.0}$	2.8 ± 1.5	2.0 ± 0.0	$\mathbf{1.9 \pm 1.4}$
4	2.0 ± 0.8	1.5 ± 0.2	$\mathbf{1.3 \pm 0.3}$	2.2 ± 0.9	$\mathbf{1.6 \pm 0.4}$	$\mathbf{1.6 \pm 0.1}$	2.6 ± 0.9	1.9 ± 0.7	$\mathbf{1.8 \pm 0.0}$	3.0 ± 1.1	2.1 ± 0.3	$\mathbf{2.0 \pm 0.9}$	3.4 ± 0.2	2.3 ± 0.1	$\mathbf{2.1 \pm 0.0}$
5	1.4 ± 0.2	0.9 ± 0.3	$\mathbf{0.8 \pm 0.2}$	1.7 ± 0.7	$\mathbf{1.0 \pm 0.5}$	$\mathbf{1.0 \pm 0.1}$	2.0 ± 1.2	$\mathbf{1.2 \pm 0.1}$	$\mathbf{1.2 \pm 0.5}$	2.3 ± 1.3	1.4 ± 0.7	$\mathbf{1.3 \pm 0.1}$	2.6 ± 0.3	1.6 ± 0.5	$\mathbf{1.5 \pm 0.4}$
6	1.4 ± 0.9	1.3 ± 0.1	$\mathbf{1.2 \pm 0.1}$	1.7 ± 0.2	$\mathbf{1.5 \pm 0.9}$	$\mathbf{1.5 \pm 0.1}$	2.0 ± 0.5	1.8 ± 0.6	$\mathbf{1.7 \pm 0.5}$	2.3 ± 1.5	2.1 ± 1.8	$\mathbf{1.9 \pm 0.5}$	2.6 ± 0.8	2.3 ± 1.0	$\mathbf{2.1 \pm 0.2}$
7	1.3 ± 0.4	1.0 ± 0.1	$\mathbf{0.9 \pm 0.5}$	1.4 ± 0.6	1.1 ± 0.3	$\mathbf{1.0 \pm 0.2}$	1.6 ± 0.2	1.3 ± 0.0	$\mathbf{1.2 \pm 0.3}$	1.8 ± 0.9	1.5 ± 0.8	$\mathbf{1.4 \pm 0.7}$	2.0 ± 0.5	1.6 ± 0.0	$\mathbf{1.5 \pm 0.1}$
8	1.3 ± 0.3	1.0 ± 0.0	$\mathbf{0.9 \pm 0.3}$	1.6 ± 0.3	1.3 ± 0.4	$\mathbf{1.1 \pm 0.4}$	1.9 ± 0.7	1.4 ± 0.1	$\mathbf{1.3 \pm 0.4}$	2.1 ± 0.3	1.6 ± 1.0	$\mathbf{1.5 \pm 0.3}$	2.3 ± 0.7	$\mathbf{1.7 \pm 0.2}$	$\mathbf{1.7 \pm 0.4}$
9	1.2 ± 0.6	0.8 ± 0.0	$\mathbf{0.7 \pm 0.1}$	1.2 ± 0.2	0.9 ± 0.2	$\mathbf{0.8 \pm 0.1}$	1.4 ± 0.1	1.0 ± 0.2	$\mathbf{0.9 \pm 0.4}$	1.5 ± 0.3	$\mathbf{1.1 \pm 0.2}$	$\mathbf{1.1 \pm 0.4}$	1.5 ± 0.8	1.2 ± 0.3	$\mathbf{1.1 \pm 0.1}$
10	1.5 ± 0.0	1.0 ± 0.3	$\mathbf{0.9 \pm 0.1}$	1.6 ± 0.4	1.1 ± 0.2	$\mathbf{1.0 \pm 0.1}$	2.0 ± 0.7	1.3 ± 0.2	$\mathbf{1.2 \pm 0.1}$	2.3 ± 1.3	1.5 ± 0.1	$\mathbf{1.4 \pm 0.1}$	2.4 ± 1.5	1.6 ± 1.2	$\mathbf{1.5 \pm 0.5}$
11	1.4 ± 0.1	1.2 ± 0.1	$\mathbf{1.1 \pm 0.7}$	1.5 ± 0.7	$\mathbf{1.3 \pm 0.1}$	$\mathbf{1.3 \pm 0.1}$	1.7 ± 0.4	1.5 ± 0.3	$\mathbf{1.4 \pm 0.7}$	1.9 ± 0.7	1.6 ± 0.1	$\mathbf{1.4 \pm 0.0}$	2.1 ± 0.5	1.7 ± 0.5	$\mathbf{1.6 \pm 0.9}$
12	2.1 ± 0.0	1.6 ± 0.8	$\mathbf{1.4 \pm 0.4}$	2.6 ± 0.0	1.9 ± 0.0	$\mathbf{1.7 \pm 0.2}$	3.1 ± 1.9	2.1 ± 0.1	$\mathbf{2.0 \pm 1.7}$	3.5 ± 1.0	2.3 ± 0.1	$\mathbf{2.2 \pm 0.0}$	3.8 ± 0.0	2.6 ± 1.3	$\mathbf{2.4 \pm 0.6}$
13	1.2 ± 0.3	0.9 ± 0.2	$\mathbf{0.8 \pm 0.0}$	1.3 ± 0.4	1.0 ± 0.4	$\mathbf{0.9 \pm 0.2}$	1.4 ± 1.1	1.1 ± 0.5	$\mathbf{1.0 \pm 0.1}$	1.6 ± 0.0	1.3 ± 0.1	$\mathbf{1.2 \pm 0.7}$	1.6 ± 0.8	1.4 ± 0.5	$\mathbf{1.3 \pm 0.2}$
14	3.1 ± 0.4	2.1 ± 0.3	$\mathbf{1.9 \pm 1.2}$	–	–	–	–	–	–	–	–	–	–	–	–
15	1.1 ± 0.3	$\mathbf{0.8 \pm 0.2}$	$\mathbf{0.8 \pm 0.3}$	1.2 ± 0.4	0.9 ± 0.1	$\mathbf{0.8 \pm 0.0}$	1.3 ± 0.1	1.0 ± 0.1	$\mathbf{0.9 \pm 0.3}$	1.5 ± 0.5	1.1 ± 0.1	$\mathbf{1.0 \pm 0.9}$	1.6 ± 0.1	$\mathbf{1.1 \pm 0.1}$	$\mathbf{1.1 \pm 0.3}$
16	3.9 ± 0.1	1.0 ± 0.2	$\mathbf{0.9 \pm 0.1}$	–	–	–	–	–	–	–	–	–	–	–	–
17	1.2 ± 0.2	$\mathbf{0.8 \pm 0.2}$	$\mathbf{0.8 \pm 0.2}$	1.4 ± 0.3	1.0 ± 0.6	$\mathbf{0.9 \pm 0.2}$	1.7 ± 0.4	1.1 ± 0.2	$\mathbf{1.0 \pm 0.7}$	1.9 ± 0.4	1.3 ± 0.1	$\mathbf{1.1 \pm 0.2}$	2.1 ± 0.3	1.4 ± 0.2	$\mathbf{1.3 \pm 0.3}$
18	2.0 ± 0.1	1.3 ± 0.0	$\mathbf{1.2 \pm 0.7}$	2.4 ± 0.7	1.5 ± 0.6	$\mathbf{1.4 \pm 0.5}$	2.9 ± 2.0	1.7 ± 0.7	$\mathbf{1.6 \pm 0.1}$	3.4 ± 1.1	1.9 ± 0.3	$\mathbf{1.7 \pm 0.8}$	3.7 ± 0.3	2.1 ± 0.5	$\mathbf{1.9 \pm 0.4}$
19	1.7 ± 0.8	$\mathbf{1.1 \pm 0.2}$	$\mathbf{1.1 \pm 0.2}$	2.0 ± 0.1	1.3 ± 0.1	$\mathbf{1.2 \pm 0.1}$	2.4 ± 1.1	1.5 ± 0.8	$\mathbf{1.4 \pm 0.8}$	2.8 ± 2.0	1.6 ± 0.1	$\mathbf{1.5 \pm 0.1}$	3.0 ± 0.5	1.8 ± 0.5	$\mathbf{1.7 \pm 0.8}$
20	1.9 ± 0.1	1.3 ± 0.2	$\mathbf{1.2 \pm 0.5}$	2.2 ± 0.1	$\mathbf{1.4 \pm 0.3}$	$\mathbf{1.4 \pm 0.3}$	2.7 ± 0.3	1.6 ± 0.5	$\mathbf{1.5 \pm 0.5}$	3.1 ± 1.6	1.8 ± 0.2	$\mathbf{1.7 \pm 0.2}$	3.3 ± 1.2	2.0 ± 0.3	$\mathbf{1.8 \pm 0.4}$
21	1.3 ± 0.2	$\mathbf{0.4 \pm 0.1}$	$\mathbf{0.4 \pm 0.0}$	1.3 ± 0.1	$\mathbf{0.4 \pm 0.1}$	$\mathbf{0.4 \pm 0.1}$	1.5 ± 0.2	$\mathbf{0.5 \pm 0.1}$	$\mathbf{0.5 \pm 0.3}$	1.7 ± 0.1	$\mathbf{0.6 \pm 0.3}$	$\mathbf{0.6 \pm 0.0}$	2.2 ± 0.6	$\mathbf{0.7 \pm 0.3}$	$\mathbf{0.7 \pm 0.3}$
22	1.5 ± 0.0	0.7 ± 0.1	$\mathbf{0.6 \pm 0.1}$	1.6 ± 0.5	0.7 ± 0.1	$\mathbf{0.6 \pm 0.2}$	1.8 ± 0.3	$\mathbf{0.8 \pm 0.7}$	$\mathbf{0.8 \pm 0.0}$	1.9 ± 0.2	$\mathbf{0.9 \pm 0.0}$	$\mathbf{0.9 \pm 0.4}$	2.2 ± 0.3	1.1 ± 0.3	$\mathbf{1.0 \pm 0.2}$
23	1.5 ± 0.1	$\mathbf{0.3 \pm 0.2}$	$\mathbf{0.3 \pm 0.1}$	1.7 ± 0.2	$\mathbf{0.4 \pm 0.1}$	$\mathbf{0.4 \pm 0.2}$	1.8 ± 0.3	$\mathbf{0.5 \pm 0.0}$	$\mathbf{0.5 \pm 0.1}$	1.8 ± 0.3	0.6 ± 0.1	$\mathbf{0.5 \pm 0.1}$	2.0 ± 0.2	$\mathbf{0.7 \pm 0.1}$	$\mathbf{0.7 \pm 0.1}$
24	1.1 ± 0.2	$\mathbf{0.5 \pm 0.2}$	$\mathbf{0.5 \pm 0.3}$	1.2 ± 0.1	$\mathbf{0.6 \pm 0.2}$	$\mathbf{0.6 \pm 0.1}$	1.2 ± 0.0	0.7 ± 0.3	$\mathbf{0.6 \pm 0.4}$	1.2 ± 0.1	$\mathbf{0.8 \pm 0.1}$	$\mathbf{0.8 \pm 0.1}$	1.3 ± 0.1	1.0 ± 0.3	$\mathbf{0.9 \pm 0.1}$
25	1.2 ± 0.7	$\mathbf{0.4 \pm 0.1}$	$\mathbf{0.4 \pm 0.2}$	1.2 ± 0.4	0.5 ± 0.0	$\mathbf{0.4 \pm 0.0}$	1.3 ± 0.3	$\mathbf{0.6 \pm 0.2}$	$\mathbf{0.6 \pm 0.1}$	1.3 ± 0.0	$\mathbf{0.7 \pm 0.1}$	$\mathbf{0.7 \pm 0.0}$	1.5 ± 0.6	0.9 ± 0.1	$\mathbf{0.8 \pm 0.3}$
26	1.9 ± 0.0	0.7 ± 0.0	$\mathbf{0.6 \pm 0.1}$	2.0 ± 0.5	$\mathbf{0.8 \pm 0.4}$	$\mathbf{0.8 \pm 0.4}$	2.4 ± 0.2	1.0 ± 0.0	$\mathbf{0.9 \pm 0.5}$	2.6 ± 0.5	1.1 ± 0.1	$\mathbf{1.0 \pm 0.5}$	3.1 ± 1.0	1.1 ± 0.0	$\mathbf{1.0 \pm 0.3}$
27	1.0 ± 0.8	$\mathbf{0.4 \pm 0.0}$	$\mathbf{0.4 \pm 0.0}$	1.1 ± 0.0	$\mathbf{0.5 \pm 0.2}$	$\mathbf{0.5 \pm 0.0}$	1.1 ± 0.7	$\mathbf{0.6 \pm 0.1}$	$\mathbf{0.6 \pm 0.2}$	1.1 ± 0.1	$\mathbf{0.7 \pm 0.1}$	$\mathbf{0.7 \pm 0.2}$	1.1 ± 0.1	$\mathbf{0.8 \pm 0.4}$	$\mathbf{0.8 \pm 0.1}$
28	1.0 ± 0.2	0.4 ± 0.1	$\mathbf{0.3 \pm 0.0}$	1.1 ± 0.2	$\mathbf{0.4 \pm 0.0}$	$\mathbf{0.4 \pm 0.1}$	1.2 ± 0.4	$\mathbf{0.5 \pm 0.3}$	$\mathbf{0.5 \pm 0.3}$	1.1 ± 0.7	$\mathbf{0.6 \pm 0.1}$	$\mathbf{0.6 \pm 0.0}$	1.2 ± 0.6	0.8 ± 0.2	$\mathbf{0.7 \pm 0.2}$
29	1.9 ± 0.5	0.7 ± 0.0	$\mathbf{0.6 \pm 0.1}$	2.0 ± 1.3	0.8 ± 0.1	$\mathbf{0.7 \pm 0.2}$	2.3 ± 1.0	0.9 ± 0.6	$\mathbf{0.8 \pm 0.3}$	2.6 ± 0.2	1.0 ± 0.3	$\mathbf{0.9 \pm 0.1}$	3.0 ± 2.0	1.0 ± 0.3	$\mathbf{0.9 \pm 0.0}$
30	1.0 ± 0.3	0.4 ± 0.2	$\mathbf{0.3 \pm 0.0}$	1.1 ± 0.1	$\mathbf{0.4 \pm 0.2}$	$\mathbf{0.4 \pm 0.0}$	1.2 ± 0.2	$\mathbf{0.5 \pm 0.1}$	$\mathbf{0.5 \pm 0.2}$	1.1 ± 0.1	0.7 ± 0.2	$\mathbf{0.6 \pm 0.1}$	1.2 ± 0.3	0.8 ± 0.1	$\mathbf{0.7 \pm 0.1}$
								...							
AAiC	3.0 ± 0.4	1.6 ± 0.3	$\mathbf{1.5 \pm 0.1}$	2.9 ± 0.6	1.7 ± 0.1	$\mathbf{1.6 \pm 0.2}$	3.2 ± 0.6	2.0 ± 0.0	$\mathbf{1.8 \pm 0.2}$	3.4 ± 1.4	2.2 ± 0.5	$\mathbf{2.1 \pm 0.2}$	3.4 ± 0.1	2.4 ± 0.7	$\mathbf{2.2 \pm 0.3}$

Table 2
ELM based and RFI prediction systems KPIs (in minutes).

j	i														
	1			2			3			4			5		
AACij	RFI	SELM	DELM	RFI	SELM	DELM	RFI	SELM	DELM	RFI	SELM	DELM	RFI	SELM	DELM
1	2.9 ± 1.6	2.3 ± 1.0	$\mathbf{2.0 \pm 0.1}$	–	–	–	–	–	–	2.2 ± 0.1	2.2 ± 0.2	$\mathbf{2.1 \pm 0.6}$	–	–	–
2	$\mathbf{0.0 \pm 0.0}$	0.1 ± 0.0	0.1 ± 0.0	–	–	–	–	–	–	2.5 ± 0.9	1.7 ± 0.9	$\mathbf{1.6 \pm 0.4}$	–	–	–
3	0.2 ± 0.1	$\mathbf{0.0 \pm 0.0}$	$\mathbf{0.0 \pm 0.0}$	–	–	–	–	–	–	2.2 ± 0.2	1.6 ± 0.2	$\mathbf{1.5 \pm 0.7}$	–	–	–
4	1.7 ± 0.2	1.5 ± 0.1	$\mathbf{1.4 \pm 0.1}$	2.3 ± 0.5	1.8 ± 0.1	$\mathbf{1.7 \pm 0.0}$	2.9 ± 1.0	1.8 ± 0.6	$\mathbf{1.7 \pm 0.3}$	–	–	–	–	–	–
5	–	–	–	1.1 ± 0.2	1.1 ± 0.6	$\mathbf{1.0 \pm 0.1}$	1.1 ± 0.6	0.9 ± 0.3	$\mathbf{0.8 \pm 0.0}$	–	–	–	–	–	–
6	–	–	–	$\mathbf{1.2 \pm 0.5}$	1.4 ± 0.1	1.3 ± 0.3	1.8 ± 0.0	1.8 ± 0.4	$\mathbf{1.7 \pm 0.9}$	–	–	–	–	–	–
7	–	–	–	1.8 ± 0.4	1.3 ± 0.3	$\mathbf{1.2 \pm 0.1}$	1.7 ± 0.1	$\mathbf{1.5 \pm 0.1}$	$\mathbf{1.5 \pm 0.4}$	–	–	–	–	–	–
8	–	–	–	1.5 ± 0.4	1.4 ± 0.2	$\mathbf{1.2 \pm 0.4}$	3.0 ± 2.7	2.5 ± 0.5	$\mathbf{2.3 \pm 0.1}$	–	–	–	–	–	–
9	–	–	–	1.1 ± 1.1	1.0 ± 0.1	$\mathbf{0.9 \pm 0.1}$	1.2 ± 0.5	$\mathbf{1.1 \pm 0.4}$	$\mathbf{1.1 \pm 0.4}$	–	–	–	–	–	–
10	–	–	–	1.9 ± 0.0	1.2 ± 0.4	$\mathbf{1.1 \pm 0.4}$	1.8 ± 0.2	1.4 ± 0.5	$\mathbf{1.2 \pm 0.1}$	–	–	–	–	–	–
11	1.3 ± 0.0	$\mathbf{1.1 \pm 0.3}$	$\mathbf{1.1 \pm 0.4}$	1.8 ± 1.3	1.1 ± 0.2	$\mathbf{1.0 \pm 0.4}$	1.2 ± 0.1	1.1 ± 0.0	$\mathbf{1.0 \pm 0.0}$	–	–	–	–	–	–
12	–	–	–	–	–	–	–	–	–	3.9 ± 0.0	1.0 ± 0.3	$\mathbf{0.9 \pm 0.1}$	–	–	–
13	–	–	–	–	–	–	–	–	–	5.8 ± 2.9	2.7 ± 0.5	$\mathbf{2.6 \pm 0.5}$	–	–	–
14	–	–	–	–	–	–	–	–	–	6.7 ± 0.6	4.3 ± 0.6	$\mathbf{4.1 \pm 0.6}$	–	–	–
15	–	–	–	–	–	–	–	–	–	3.8 ± 0.3	1.0 ± 0.3	$\mathbf{0.9 \pm 0.0}$	–	–	–
16	–	–	–	–	–	–	–	–	–	3.7 ± 2.4	1.0 ± 0.1	$\mathbf{0.9 \pm 0.2}$	–	–	–
17	–	–	–	–	–	–	–	–	–	5.9 ± 2.6	2.4 ± 0.1	$\mathbf{2.2 \pm 0.0}$	–	–	–
18	–	–	–	–	–	–	–	–	–	–	–	–	$\mathbf{1.6 \pm 0.3}$	1.9 ± 0.9	1.8 ± 0.3
19	–	–	–	–	–	–	–	–	–	–	–	–	1.1 ± 0.3	1.1 ± 0.1	$\mathbf{1.0 \pm 0.3}$
20	–	–	–	–	–	–	–	–	–	–	–	–	2.5 ± 0.4	2.3 ± 0.8	$\mathbf{2.1 \pm 0.6}$
21	–	–	–	–	–	–	–	–	–	–	–	–	1.3 ± 0.2	1.3 ± 0.5	$\mathbf{1.2 \pm 0.3}$
22	–	–	–	–	–	–	–	–	–	–	–	–	2.5 ± 0.4	2.3 ± 1.2	$\mathbf{2.2 \pm 0.7}$
23	–	–	–	–	–	–	–	–	–	–	–	–	2.4 ± 0.4	2.2 ± 0.8	$\mathbf{1.9 \pm 0.3}$
24	–	–	–	1.3 ± 0.3	$\mathbf{0.9 \pm 0.1}$	$\mathbf{0.9 \pm 0.4}$	1.2 ± 0.0	0.8 ± 0.1	$\mathbf{0.7 \pm 0.2}$	–	–	–	–	–	–
25	–	–	–	2.5 ± 1.4	1.4 ± 0.5	$\mathbf{1.3 \pm 0.6}$	1.9 ± 0.5	0.9 ± 0.5	$\mathbf{0.8 \pm 0.1}$	–	–	–	–	–	–
26	–	–	–	1.6 ± 0.1	1.3 ± 0.1	$\mathbf{1.2 \pm 0.1}$	1.6 ± 0.7	1.1 ± 0.2	$\mathbf{1.0 \pm 0.4}$	–	–	–	–	–	–
27	–	–	–	1.2 ± 0.6	0.9 ± 0.0	$\mathbf{0.8 \pm 0.1}$	1.3 ± 0.1	1.1 ± 0.1	$\mathbf{1.0 \pm 0.2}$	–	–	–	–	–	–
28	–	–	–	1.5 ± 0.3	1.1 ± 0.2	$\mathbf{1.0 \pm 0.1}$	1.7 ± 0.7	0.9 ± 0.0	$\mathbf{0.8 \pm 0.1}$	–	–	–	–	–	–
29	–	–	–	1.3 ± 0.4	$\mathbf{1.1 \pm 0.2}$	$\mathbf{1.1 \pm 0.8}$	1.4 ± 0.5	1.2 ± 0.2	$\mathbf{1.1 \pm 0.1}$	–	–	–	–	–	–
30	–	–	–	2.5 ± 0.5	2.1 ± 0.4	$\mathbf{2.0 \pm 0.4}$	2.1 ± 1.0	1.8 ± 0.5	$\mathbf{1.7 \pm 0.2}$	–	–	–	–	–	–
								...							
AACi	3.3 ± 0.1	1.5 ± 0.3	$\mathbf{1.4 \pm 0.4}$	3.1 ± 1.6	1.5 ± 0.0	$\mathbf{1.3 \pm 0.3}$	3.3 ± 0.5	1.4 ± 0.0	$\mathbf{1.3 \pm 0.4}$	4.2 ± 0.6	2.2 ± 0.8	$\mathbf{2.1 \pm 0.3}$	6.2 ± 2.4	4.2 ± 1.6	$\mathbf{3.9 \pm 1.6}$

Table 3
ELM based and RFI prediction systems KPIs (in minutes).

j	TAAj		
	RFI	SELM	DELM
1	2.2 ± 0.8	1.9 ± 0.9	$\mathbf{1.7 \pm 0.0}$
2	4.3 ± 0.2	2.1 ± 0.5	$\mathbf{2.0 \pm 1.7}$
3	2.3 ± 0.5	1.5 ± 0.5	$\mathbf{1.4 \pm 0.2}$
4	2.4 ± 0.5	1.7 ± 0.4	$\mathbf{1.5 \pm 0.5}$
5	1.7 ± 0.4	1.1 ± 0.3	$\mathbf{1.0 \pm 0.5}$
6	1.9 ± 0.3	$\mathbf{1.6 \pm 0.3}$	$\mathbf{1.6 \pm 0.1}$
7	1.5 ± 0.7	1.2 ± 0.2	$\mathbf{1.1 \pm 0.6}$
8	1.9 ± 0.5	$\mathbf{1.4 \pm 0.3}$	$\mathbf{1.4 \pm 0.3}$
9	1.4 ± 0.5	0.9 ± 0.2	$\mathbf{0.8 \pm 0.2}$
10	1.8 ± 0.1	1.1 ± 0.3	$\mathbf{1.0 \pm 0.8}$
11	1.8 ± 0.6	1.4 ± 1.0	$\mathbf{1.3 \pm 0.0}$
12	2.8 ± 2.2	1.9 ± 0.5	$\mathbf{1.7 \pm 0.5}$
13	1.4 ± 0.4	1.1 ± 0.1	$\mathbf{1.0 \pm 0.2}$
14	3.1 ± 0.6	2.0 ± 0.1	$\mathbf{1.8 \pm 0.9}$
15	1.2 ± 0.5	0.9 ± 0.3	$\mathbf{0.8 \pm 0.1}$
16	3.9 ± 0.5	$\mathbf{0.9 \pm 0.2}$	$\mathbf{0.9 \pm 0.3}$
17	5.8 ± 1.4	2.6 ± 0.6	$\mathbf{2.5 \pm 0.9}$
18	6.7 ± 2.5	4.1 ± 0.4	$\mathbf{3.8 \pm 0.7}$
19	3.8 ± 0.7	$\mathbf{0.9 \pm 0.1}$	$\mathbf{0.9 \pm 0.4}$
20	3.7 ± 1.1	$\mathbf{0.9 \pm 0.2}$	$\mathbf{0.9 \pm 0.1}$
21	5.9 ± 0.8	2.3 ± 1.3	$\mathbf{2.1 \pm 0.5}$
22	4.9 ± 0.7	2.1 ± 0.4	$\mathbf{2.0 \pm 0.2}$
23	6.5 ± 1.1	3.5 ± 2.1	$\mathbf{3.2 \pm 0.3}$
24	5.1 ± 0.9	$\mathbf{2.1 \pm 0.3}$	$\mathbf{2.1 \pm 0.2}$
25	4.6 ± 1.4	1.7 ± 0.0	$\mathbf{1.6 \pm 0.6}$
26	5.6 ± 1.8	2.7 ± 0.2	$\mathbf{2.6 \pm 1.0}$
27	6.2 ± 0.1	2.7 ± 0.6	$\mathbf{2.4 \pm 0.3}$
28	5.5 ± 3.6	2.6 ± 1.8	$\mathbf{2.4 \pm 0.7}$
29	4.2 ± 0.8	$\mathbf{1.0 \pm 0.4}$	$\mathbf{1.0 \pm 0.4}$
30	4.7 ± 1.0	1.7 ± 0.4	$\mathbf{1.6 \pm 0.8}$
...			
TAA	3.3 ± 1.8	1.9 ± 0.6	$\mathbf{1.7 \pm 0.8}$

particularly by the RFI operators, which can take better decisions. For example, few minutes can change the point where a train has to overtake another one of many kilometers, because many line sections do not allow to perform this maneuver. Finally, note that the accuracy decreases as j increases, since the forecasts refer to events which are further into the future. Moreover, since some trains have less checkpoints than the others, a symbol '–' has been placed for those checkpoints that are not included in the itinerary of the considered trains (see for example train $j = 14$, which only passes through two checkpoints).

- Table 2 reports the AACij and the AACi. From Table 2 it is possible to derive the same observations derived from Table 1. In this case, it is also important to underline that not all the trains run over all the checkpoints, and this is the reason why for some combinations of train j and station i there is a symbol '–'.
- Table 3 reports the TAAj and the TAA. The latter is more concise and underlines better the advantage, from a final performance perspective, of the DELM with respect to the actual RFI prediction system.

In Table 4 we report, both for SELM and DELM, the mean optimal values of the different hyperparameters, averaged over the different TDPS models. Table 4 suggests and gives insight over the optimal average TDPS model architecture. Note that, for DELM, a deep architecture with small amount of neurons in each layer is preferred with respect to the SELM architecture which needs much more hidden neurons. Moreover, it is possible to observe that the DELM architecture tries, on average, to distill the information layerwise since h_i decreases with i.

Finally, we compared the performance of a Matlab and Apache Spark implementations of the training phase of SELM and DELM.

Table 4
SELM and DELM average optimal values of the different hyperparameters.

Architecture	Hyperparameter	Average value
SELM	h	5145 ± 3233
DELM	l	6 ± 3
	h_1	685 ± 986
	h_2	589 ± 822
	h_3	545 ± 856
	h_4	451 ± 743
	h_5	412 ± 732
	h_6	402 ± 644
	h_7	421 ± 640
	h_8	367 ± 439
	h_9	234 ± 139
	h_{10}	112 ± 56

The first implementation run on a PC with 4 Intel Xeon CPU E5-4620@2.20 GHz, 128 GB of RAM, 500 GB of SSD running Windows Server 2012 R2 and Matlab R2016a. The second one, instead, run over four n1-standard-16 machines of the Google Compute Engine, which include 60 GB of ram, 16 cores and 500 GB SSD disk each, allowing the deployment of a cluster with Spark 1.6.2 and Hadoop 2.6.4, check our previous paper [42] for more details. In order to perform the experiments reported in the paper, the Matlab implementation did not finished either 1% of the experiments after 1 month, while our spark implementation took approximately one day.

6. Conclusions

This paper deals with the problem of building a TDPS based on state-of-the-art tools and techniques able to rapidly grasp the knowledge hidden in historical data about TM. In particular, the proposed solution improves the state-of-the-art methodologies actually exploited from the IM like RFI. Results on real world TM data provided by RFI show that advanced analytics approaches can perform up to twice better than current state-of-the-art methodologies. In particular, exploiting historical data about TM gives robust models with high performance with respect to the actual TD prediction system of RFI. We have also shown how to efficiently and effectively tune the hyperparameters involved in the learning algorithms. Finally, by exploiting the Apache Spark in memory technology, we have been able to build a system with high performance, also in terms of the required training time for building all the models needed for dealing with a large-scale Railway Network. Future works will take into account also exogenous information available from external sources, such as weather information, information about passenger flows by using touristic databases, about railway assets conditions, or any other source of data which may affect railway dispatching operations.

Acknowledgements

This research has been supported by the European Union through the projects Capacity for Rail – C4R (European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement 605650) and Innovative Intelligent Rail – In2Rail (European Union's Horizon 2020 research and innovation programme under grant agreement 635900).

References

[1] A. Thaduri, D. Galar, U. Kumar, Railway assets: a potential domain for big data analytics, in: The INNS Big Data Conference, 2015.
[2] A.M. Zarembski, Some examples of big data in railroad engineering, in: IEEE International Conference on Big Data, 2014.

[3] A.Y. Al-Jarrah, P.D. Yoo, S. Muhaidat, G.K. Karagiannidis, K. Taha, Efficient machine learning for big data: a review, Big Data Res. 2 (3) (2015) 87–93.
[4] E. Fumeo, L. Oneto, D. Anguita, Condition based maintenance in railway transportation systems based on big data streaming analysis, in: The INNS Big Data Conference, 2015.
[5] H. Li, D. Parikh, Q. He, B. Qian, Z. Li, D. Fang, A. Hampapur, Improving rail network velocity: a machine learning approach to predictive maintenance, Transp. Res., Part C, Emerg. Technol. 45 (2014) 17–26.
[6] A. Núñez, J. Hendriks, Z. Li, B. De Schutter, R. Dollevoet, Facilitating maintenance decisions on the dutch railways using big data: the ABA case study, in: IEEE International Conference on Big Data, 2014.
[7] H. Feng, Z. Jiang, F. Xie, P. Yang, J. Shi, L. Chen, Automatic fastener classification and defect detection in vision-based railway inspection systems, IEEE Trans. Instrum. Meas. 63 (4) (2014) 877–888.
[8] C. Aytekin, Y. Rezaeitabar, S. Dogru, I. Ulusoy, Railway fastener inspection by real-time machine vision, IEEE Trans. Syst. Man Cybern. Syst. 45 (7) (2015) 1101–1107.
[9] M. Figueres-Esteban, P. Hughes, C. Van Gulijk, The role of data visualization in railway big data risk analysis, in: European Safety and Reliability Conference, 2015.
[10] S.A. Branishtov, Y.A. Vershinin, D.A. Tumchenok, A.M. Shirvanyan, Graph methods for estimation of railway capacity, in: International Conference on Intelligent Transportation Systems, 2014.
[11] Y. Bai, T.K. Ho, B. Mao, Y. Ding, S. Chen, Energy-efficient locomotive operation for Chinese mainline railways by fuzzy predictive control, IEEE Trans. Intell. Transp. Syst. 15 (3) (2014) 938–948.
[12] Z. Xueyan, G. Depeng, Application of big data technology in marketing decisions for railway freight, in: The International Conference of Logistics Engineering and Management, 2014.
[13] J. Tutcher, Ontology-driven data integration for railway asset monitoring applications, in: IEEE International Conference on Big Data, 2014.
[14] Q. Li, Z. Zhong, Z. Liang, Y. Liang, Rail inspection meets big data: methods and trends, in: 18th International Conference on Network-Based Information Systems, 2015.
[15] M. Ma, P. Wang, C.H. Chu, L. Liu, Efficient multipattern event processing over high-speed train data streams, IEEE Int. Things J. 2 (4) (2015) 295–309.
[16] F. Wang, T.h. Xu, Y. Zhao, Y.r. Huang, Prior LDA and SVM based fault diagnosis of vehicle on-board equipment for high speed railway, in: IEEE 18th International Conference on Intelligent Transportation Systems, 2015.
[17] Y. Zhao, T.h. Xu, W. Hai-feng, Text mining based fault diagnosis of vehicle onboard equipment for high speed railway, in: 17th IEEE International Conference on Intelligent Transportation Systems, 2014.
[18] K. Noori, K. Jenab, Fuzzy reliability-based traction control model for intelligent transportation systems, IEEE Trans. Syst. Man Cybern. Syst. 43 (1) (2013) 229–234.
[19] Z. Bin, X. Wensheng, An improved algorithm for high speed train's maintenance data mining based on mapreduce, in: International Conference on Cloud Computing and Big Data, 2015.
[20] B. Wang, F. Li, X. Hei, W. Ma, L. Yu, Research on storage and retrieval method of mass data for high-speed train, in: International Conference on Computational Intelligence and Security, 2015.
[21] J. Sadler, D. Griffin, A. Gilchrist, J. Austin, O. Kit, J. Heavisides, GeoSRM – online geospatial safety risk model for the GB rail network, IET Intell. Transp. Syst. 10 (1) (2016) 17–24.
[22] Y. Qingyang, Y. Xiaoyun, Scheduling optimization model and algorithm design for two-way marshalling train, in: International Conference on Intelligent Transportation, Big Data and Smart City, 2015.
[23] Y.T. Zhu, F.Z. Wang, X.H. Shan, X.Y. Lv, K-medoids clustering based on mapreduce and optimal search of medoids, in: International Conference on Computer Science Education, 2014.
[24] E. Davey, Rail traffic management systems (TMS), in: IET Professional Development Course Railway Signalling and Control Systems, 2012.
[25] M. Dotoli, N. Epicoco, M. Falagario, C. Seatzu, B. Turchiano, A decision support system for optimizing operations at intermodal railroad terminals, IEEE Trans. Syst. Man Cybern. Syst. (2016) 1–15.
[26] J.F. Cordeau, P. Toth, D. Vigo, A survey of optimization models for train routing and scheduling, Transp. Sci. 32 (4) (1998) 380–404.
[27] T. Dollevoet, F. Corman, A. D'Ariano, D. Huisman, An iterative optimization framework for delay management and train scheduling, Flex. Serv. Manuf. J. 26 (4) (2014) 490–515.
[28] S. Milinković, M. Marković, S. Vesković, M. Ivić, N. Pavlović, A fuzzy Petri net model to estimate train delays, Simul. Model. Pract. Theory 33 (2013) 144–157.
[29] A. Berger, A. Gebhardt, M. Müller-Hannemann, M. Ostrowski, Stochastic delay prediction in large train networks, in: OASIcs – OpenAccess Series in Informatics, 2011.
[30] S. Pongnumkul, T. Pechprasarn, N. Kunaseth, K. Chaipah, Improving arrival time prediction of Thailand's passenger trains using historical travel times, in: International Joint Conference on Computer Science and Software Engineering, 2014.
[31] R.M.P. Goverde, A delay propagation algorithm for large-scale railway traffic networks, Transp. Res., Part C, Emerg. Technol. 18 (3) (2010) 269–287.
[32] I.A. Hansen, R.M.P. Goverde, D.J. Van Der Meer, Online train delay recognition and running time prediction, in: IEEE International Conference on Intelligent Transportation Systems, 2010.
[33] P. Kecman, Models for Predictive Railway Traffic Management, PhD Thesis, TU Delft, Delft University of Technology, 2014.
[34] P. Kecman, R.M.P. Goverde, Online data-driven adaptive prediction of train event times, IEEE Trans. Intell. Transp. Syst. 16 (1) (2015) 465–474.
[35] G. Huang, G.B. Huang, S. Song, K. You, Trends in extreme learning machines: a review, Neural Netw. 61 (2015) 32–48.
[36] L.L.C. Kasun, H. Zhou, G.B. Huang, C.M. Vong, Representational learning with elms for big data, IEEE Intell. Syst. 28 (6) (2013) 31–34.
[37] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Cogn. Model. 5 (3) (1988) 1.
[38] Y. Bengio, A. Courville, P. Vincent, Representation learning: a review and new perspectives, IEEE Trans. Pattern Anal. Mach. Intell. 35 (8) (2013) 1798–1828.
[39] J. Tang, C. Deng, G.B. Huang, Extreme learning machine for multilayer perceptron, IEEE Trans. Neural Netw. Learn. Syst. 27 (4) (2016) 809–821.
[40] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing, in: USENIX Conference on Networked Systems Design and Implementation, 2012.
[41] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D.B.T. Manish Amde, S. Owen, D. Xin, R. Xin, M.J. Franklin, R. Zadeh, M. Zaharia, A. Talwalkar, MLlib: machine learning in apache spark, J. Mach. Learn. Res. 17 (34) (2016) 1–7.
[42] J.L. Reyes-Ortiz, L. Oneto, D. Anguita, Big data analytics in the cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf, in: The INNS Big Data Conference, 2015.
[43] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[44] T. White, Hadoop: The Definitive Guide, O'Reilly, Media, 2012.
[45] Google, Google Compute Engine, https://cloud.google.com/compute/, 2016, online; accessed 3 May 2016.
[46] Rete Ferroviaria Italiana, Gruppo Ferrovie Dello Stato Italiane, http://www.rfi.it/, 2016, online; accessed 3 May 2016.
[47] F. Takens, Detecting Strange Attractors in Turbulence, Springer, 1981.
[48] V.N. Vapnik, Statistical Learning Theory, Wiley, New York, 1998.
[49] R. Caruana, S. Lawrence, G. Lee, Overfitting in neural nets: backpropagation, conjugate gradient, and early stopping, in: Neural Information Processing Systems, 2001.
[50] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.
[51] D. Anguita, A. Ghio, L. Oneto, S. Ridella, In-sample and out-of-sample model selection and error estimation for support vector machines, IEEE Trans. Neural Netw. Learn. Syst. 23 (9) (2012) 1390–1406.
[52] S. Floyd, M. Warmuth, Sample compression, learnability, and the Vapnik–Chervonenkis dimension, Mach. Learn. 21 (3) (1995) 269–304.
[53] L. Oneto, A. Ghio, S. Ridella, D. Anguita, Fully empirical and data-dependent stability-based bounds, IEEE Trans. Cybern. 45 (9) (2015) 1913–1926.
[54] P. Germain, A. Lacasse, F. Laviolette, M. Marchand, J.-F. Roy, Risk bounds for the majority vote: from a PAC-Bayesian analysis to a learning algorithm, J. Mach. Learn. Res. 16 (4) (2015) 787–860.
[55] C. Dwork, V. Feldman, M. Hardt, T. Pitassi, O. Reingold, A. Roth, The reusable holdout: preserving validity in adaptive data analysis, Science 349 (6248) (2015) 636–638.
[56] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: International Joint Conference on Artificial Intelligence, 1995.
[57] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (2012) 281–305.

Please cite this article in press as: L. Oneto et al., Train Delay Prediction Systems: A Big Data Analytics Perspective, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.05.002