Traffic Event Prediction based on Machine Learning and Complex Event Processing

Huaicheng Zhang_250963351
Department of Electrical and Computer Engineering
The University of Western Ontario, London, Canada
Email: hzhan723@uwo.ca

Abstract — **In the sphere of future intelligent transportation systems and management of road traffic, the concept of traffic event prediction is identified as a pivotal component capable of playing a crucial role. The need for recognizing patterns in near real-time has stimulated the development of complex event processing (CEP) and other related areas. CEP provides a distribution solution for analyzing data flows in intelligent transportation system (ITS) applications, but it only acts on real-time data and fails to make prediction by learning historical data. In this paper, an innovative system combining machine learning and CEP is proposed to solve this dilemma. For machine learning component, we also propose an adaptive prediction algorithm which controls the error by using dynamic prediction window. The ideal range for prediction window is provided by Q-learning based on prediction error and processing time. Evaluation results on different real-world traffic scenarios demonstrate that the algorithm can provide predictions with an accuracy of 95%. The method of error modeling based on t distribution is exploited to improve the system performance as well. Our integrated method provides satisfactory performance on chosen traffic prediction case and can further be applied to predict complex events for different domains.**

Keywords — **Intelligent transportation system, complex event processing, machine learning, data streams, traffic prediction.**

I. INTRODUCTION

Prediction on road traffic has become a topic of great concern in recent years with the popularization of intelligent systems. Data generated by devices ranging from road cameras to indoor sensors provides more opportunities and available support for these intelligent applications. Intelligent transportation system (ITS) is an emerging field because of this benefit, where information technology, network and electronics are used to improve the transportation efficiency.

Traffic event such as congestion is an important element in the research area of ITS. With the steadily increasing number of vehicles, traffic congestion has become an inevitable problem and predicting road congestion in advance has become more meaningful than before. Providing accurate and predictive traffic conditions to the participants of transportation systems, especially the drivers, would be a promising and necessary solution to this problem. Compared to periods when traffic data from traditional sensors and traffic management center was not within easy

reach, nowadays online data are much easier to be acquired by individuals with the assistance of Internet and public service. The enrichment of data is significantly motivated by the increasing number of devices connected to the Internet, from 8.7 billion in 2012 to 23 billion in 2018 [1]. And the open data could be utilized for decision support systems to make timely and suitable decisions according to the real-time traffic conditions.

Traffic flow prediction and traffic event detection are both key functional components in ITS [2][3]. The traffic prediction methods are enriched with the availability of open data and the development of social media. However, data from different sources is produced and presented as real-time events or complex patterns in ITS. These events or patterns need to be processed with least amount of delay in order to put them into practice for current conditions. So complex event processing (CEP) is exploited here to satisfy the need for analyzing, processing and forecasting from complex patterns in near real-time. The inherent nature of distribution [4] makes CEP an excellent option for numerous applications in ITS including traffic event prediction. Besides, it is a desirable choice where correlating data streams from various gathering locations are used to predict more complex events for real-time analysis.

Most of CEP applications are used to find coping solutions by correlating data flows with previously defined rules while neglecting the historical data due to memory limitation. But in some practical scenarios, predicting a forthcoming event is more beneficial than discovering it after its occurrence. The importance of prediction is stressed in more related researches because as Fig. 1 shows, the value of an event gradually decreases as time goes by. Meanwhile, making prediction before the event is more difficult compared to learning after the event, which reminds us to consider both timeliness and accuracy. Traffic congestion prediction is an example as traffic management center can take preventive measures if the congestion is predicted in advance. The lack of predictive capability for CEP may be offered by machine learning (ML) algorithms and some statistical analysis methods.

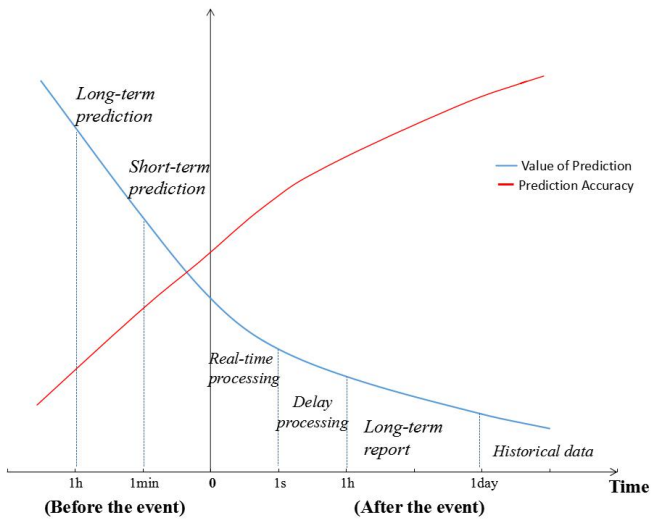


Fig. 1. Value of the event versus prediction accuracy

On the other hand, systems based on machine learning and statistical methods provide predictive answers in different application areas. However, these methods are not always

appropriate for processing data streams from different domains in real-time because they need historical data for model training. They lack the ability of expansibility and processing various streams at the same time, where CEP could be used to fill the vacancy.

In this paper, we propose a generic system integrating CEP and machine learning based on open source components and implement it to predict traffic events for proactive ITS applications. Then we improve the current machine learning model and propose an adaptive algorithm called adaptive dynamic window learning (ADWL) for traffic data prediction. The optimum range of prediction horizon is learned by Q-learning process with consideration of error and time cost. The experiments show that the proposed model tracks actual data accurately and outperforms common ML models. Besides, the results indicate that our improved algorithm reduces the prediction error by more than 40% and spends similar processing time compared with conventional ML algorithm.

The rest of the paper is organized as follows. Section II introduces the background of CEP and lists work in traffic prediction. Section III illustrates several phases of our method and the tools we utilize. Section IV explains the main algorithms used in the model and shows some details of implementation. Then we present the evaluation results in Section V and summarize our work and future direction in Section VI.

II. RELATED WORK

In previous research, CEP and ML were investigated in-depth as respective directions and were mainly aimed for different kinds of applications. CEP has been designed for processing near real-time data and linking multiple data streams in high speed without storing them. Whereas in [5] machine learning algorithms are applied to serve for specific target based on historical data for extraction of knowledge. As the increasing and miscellaneous requirements for processing IoT data stream in recent years, more hybrid approaches have been studied where predictive analytic (PA) methods based on machine learning are combined with CEP in order to provide self-motivated solutions. This idea was initially proposed in [6] where the authors presented a conceptual framework combining CEP with PA to serve as the basis of future generic design. The innovative approach led to positive discussion but they did not verify their idea on practical applications. Then the authors of [7] suggested using stochastic event processing network for event detection and then training machine learning model with complex events to predict upcoming events. It was another example of utilizing both CEP and machine learning, but the overall calculation complexity and cost of iterative optimization process raises exponentially as the training data set increases. As a result, this approach was inappropriate to large-scale IoT applications and unlikely to work effectively in a real-world case.

In literature, conventional machine learning methods have been deployed for various real-time prediction scenarios like trajectory prediction of sea vessels [8] or vehicles in real-time. Traditional algorithms consist of linear regression, decision tree, random forest, neural network and support vector machine (SVM), and each of them has its advantages when addressing different problems. Authors in [9] implemented neural network to handle traffic prediction tasks. They demonstrated the approach on collected real-time data and trained the neural network

using 60 days of data. But a major weakness of this method is the model parameters are static once after training. The statistical feature and condition of data base may change over time and the system is unable to accommodate these changes. Meanwhile, errors are possible to propagate and keep increasing influencing the reliability of the whole system. Random forest algorithm was used in [10] to predict disease like diabetes. The method obtained results with the most accurate prediction for this case but it was incapable to deal with extreme values and rapid changes. Ren *et al.* [11] analyzed temporal and spatial patterns of traffic speed bands and predicted future traffic based on support vector machine model. The proposed method showed desirable performance in short-time prediction but failed to maintain this standard when applied to prediction for next 1 hour.

In recent researches, more improvements have been made on traditional machine learning algorithms and more creative approaches were proposed to acquire satisfactory outcomes. Traffic conditions collected from web based map services were used to demonstrate the feasibility in [12]. A new method called long short-term memory (LSTM) model based on recurrent neural network was presented to learn the patterns in traffic condition sequences. The deep learning approach with stacked LSTM model outperformed decision tree model and SVM model in the evaluation. The advantages of extreme learning machine (ELM) on finding complex nonlinear relationship and learning speed were illustrated in [13]. A new prediction method called Ensemble Real-time Sequential Extreme Learning Machine (ERS-ELM) was then proposed to fit in non-stationary traffic scenarios and overcome some disadvantages of neural network. This method used both historical data and current data to predict near future situation one-by-one with highly effective learning mechanism but still needed improvements on dealing with large missing traffic flow data. Time series analysis using auto-regressive integrated moving average (ARIMA) [14] model is considered as one of the most reliable methods. Kumar *et al.* [15] studied the previous algorithm and found the seasonal autoregressive integrated moving average (SARIMA) model performed better than some conventional machine learning algorithms. Their research tried to overcome computational time and resource issues by using SARIMA model for short time prediction with only limited input data. These newly proposed methods mainly concentrate on the time correlation of traffic flow at a particular site and receive promising prediction results. However, some of them usually presume linear trends of traffic flow data which is not applicable in real traffic circumstances due to the complex spatiotemporal interactions between different flows. Moreover, these drawbacks may restrict the usage of prediction methods in practical situations.

III. PROPOSED SYSTEM MODEL

The proposed framework is illustrated in Fig. 2, where the whole architecture could be divided into three parts: preparation phase, prediction phase and event generation phase. In our approach, several aspects like combination between machine learning and CEP component, reliability and applicability of the model, and selecting a common exchange format across the whole architecture are considered.

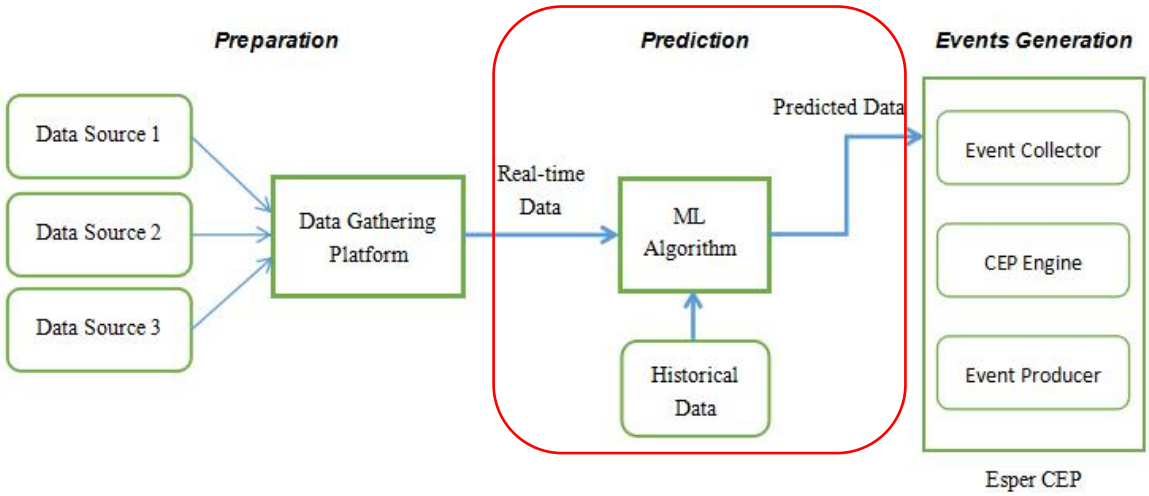


Fig. 2. Block diagram of the whole intelligent system

Preparation is the first section in our proposed method where Node-RED would be used as data gathering platform. After receiving the data stream, this component tends to perform preprocessing tasks to filter redundant data and transform the file format. Then the real-time data is delivered to our main focused section, where we apply different machine learning algorithms including proposed ADWL to complete the prediction task. Here historical data is used to train the ML component and yield more accurate predictive results combined with real-time data. The predicted data would be returned to event generation section once machine learning process is finished. In the third section, a CEP engine called Esper is exploited to collect the newly arriving events, complete the pattern matching assignment and finally generate complex events we need.

In this section, a brief introduction of the open source tools is given first. More information on the three phases of the system model and related details are presented in the following part.

A. Tools Used

1) Node-RED Tool

Node-RED is an open source visual tool which provides application programming interfaces (API) for connecting different components. In the first phase of our system, we use this tool as the data gathering and processing platform. It can also be used to filter redundant data and complete format transforming with given Java code or user defined functions.

For the first time and with the help of IoT, researchers or individuals gain the access to multifarious data sources in our real world. The data format is not limited in any particular format as IoT data sources can be MQTT data feed, RESTful Web service or any other external source. As a result, data streams from different sources may be stored in different formats and contain redundant parts that need to be filtered out.

As we know, JSON and XML are two commonly used formats especially when transmitting data feeds through ITS network. But usually files in these formats are not convenient for data analysis and data in CSV format is required. In our study, transformation between these data formats is the first task and Node-RED provides necessary help.

2) Esper CEP tool

Esper is designed for latency sensitive application such as credit fraud detection and business trading market. It is proper to be used to process real-time vehicle traffic flow which involves large volumes of streaming data. The open source condition of Esper makes it an ideal candidate for our system out of several CEP platforms in the market. Besides, Esper is our preferred choice because its mature Java-based internal structure serves CEP functionality with high throughput.

Contrary to other data stream processing applications, Esper stores queries instead of storing data and runs data when receiving a query command. The queries are written in event processing language (EPL) which is a language similar to SQL with GET, SELECT FROM, DEFINE clauses and so on. In Esper, streaming data replaces tables and events as the basic element of data and can support functions like grouping, filtering and joins over single event or set of events. Events in Esper are represented as plain old Java objects (POJO) and the output complex events detected from EPL contents are returned in the same format.

B. Preparation

In real-world environment, original road traffic data is collected by roadside detectors including microwave sensors, radar sensors and video cameras. Then the data including speed and traffic intensity is sorted out and published on the web pages.

At the beginning of the preparation phase, the Node-RED platform is formed as in Fig. 3. The raw data published through HTTP service is in XML format. After parsing the data from web page, transformation would be implemented and XML file is converted to JSON format and then into CSV format. The processed data is transmitted through a function, which is used to complete filtering task and could be defined by the user. At the end of this workspace, data would be written in a specified csv file and the data flow could be monitored by the user from debug module. The whole platform is able to retrieve the real-time data streams and can be used to store them as historical data.

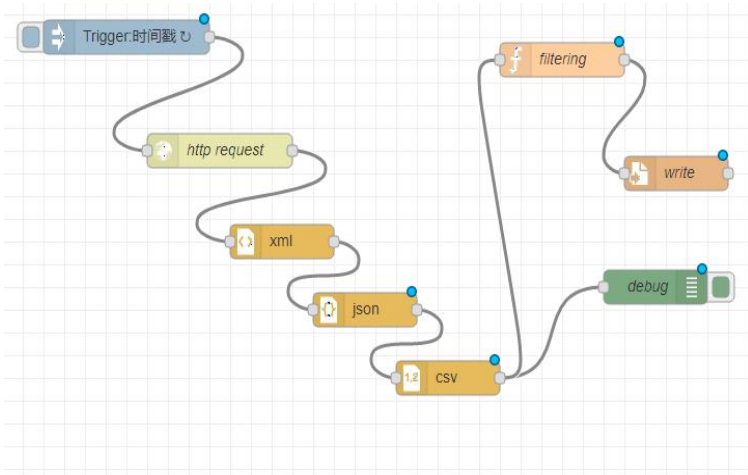


Fig. 3. Process deployment in Node-RED

C. Prediction

The prediction phase mainly consists of task of selecting machine learning algorithm and choosing the prediction window size. Selection of underlying algorithm is the first step and should be determined according to the specific application case.

There are multiple kinds of machine learning algorithms available for regression and prediction. Linear regression, decision tree, random forest, support vector machine and artificial neural network are several commonly used methods in real-world situations. Normally, more statistical methods such as ARIMA mentioned earlier are used for time series training, but support vector machine and neural networks are preferred recently in non-linear regression problems due to their ability to provide higher accuracy and the stability of the whole model.

More researches in recent times focused on the improvements of SVM and proved that combined model showed better performance than single SVM model. The main function of SVM is obtaining the optimal hyper plane such that the maximum distance between the margin and training data is reached. The model can be used for generalize unseen training data once the optimal hyper plane is obtained. For traffic flow prediction which is a non-linear problem, support vector regression (SVR) is a desirable solution because of its kernel function and is selected in our approach. The kernel plays a vital role in SVR and can solve the prediction problem of mapping it from low dimension into a high dimensional feature space. SVR can be regarded as an extension of previous model which is widely used for regression analytic [16]. Experiments in [17] show that SVR based model works well for unknown data which clearly indicates its applicability for real-time traffic flow prediction. In our approach, we use small training window so as to add less complexity to the regression algorithm. As for kernel function, we made comparison among several variants of SVR and chose the widely used radial basis function (RBF). For example, the Gaussian RBF kernel on two samples x and y with width σ is expressed as :

$$k(x, y) = \exp(-\frac{\|x - y\|^2}{2\sigma^2}) \tag{1}$$

D. Event Generation

In the event generation phase, Esper is the CEP engine we use to execute the process. We will mainly introduce this engine and error propagation in CEP framework.

As shown in our model, Esper CEP platform is more specifically divided into three connected components. Event collector serves as the front end and is responsible for retrieving data streams from different sources including MQTT or any other formats of RESTful web service. In our experiments, the reading process is prescribed in JSON format from output predicted data. After converting data to required POJO format for Esper, the processed data would be forwarded to CEP engine which is the core of event generation structure. CEP engine detects complex events by matching patterns based on predefined rules and EPL statements. Finally, detected events are delivered to the relevant applications in required data format in accordance with distribution results in event producer.

In our approach, each generated event is accompanied by prediction error and these events are forwarded to the CEP engine. The engine then applies user defined CEP rules to detect and correlate these events. Prediction error would add more chanciness in event processing and as a result influencing the dependability of the whole system. Taking prediction error into account, we propose to adopt a probabilistic event processing method to define event tuples and use PDF_E to represent the probability density function of prediction error [18].

There are several common pattern matching rules in CEP such as filtering, joins and window, and we will introduce how prediction error is propagated when applying these rules. First, event filtering is a frequently used functionality in CEP. We might receive piles of events but only interested in several specific events, then typical conditions or requirements are needed to narrow the scope of results. If we consider an example of road speed sensor which monitors the velocity of passing vehicles. When a user only concentrates on the velocities higher than a fixed threshold, the filter can be defined. And only if the conditions are met, the result would be produced and shown to the administrator. In cases where the rule is adopted to an individual event, the total error of predicting the complex event is the same as the probability of prediction error for the specified data source. For example, if an event is generated when the data stream D_1 is less than a certain standard, the final error would be expressed as:

$$\mathrm{PDF}_{\mathrm{total}} = \mathrm{PDF}_E(D_1) \tag{2}$$

Joins has the functionality of associating events from different data streams with simple logical operations such as conjunction and disjunction.

In conjunction case, the overall probability of prediction error is calculated by multiplying error probability related to each data stream. Let's assume there are separate data streams D_1 and D_2, a complex event is generated if D_1 is greater than threshold T and D_2 is less than T. Then the total error is shown as Equation (3). Condition is similar in disjunction where a complex event is triggered by satisfying either of the multiple requirements. For instance, if the rule of defining a complex event is either data D_1 or D_2 is greater than threshold T, the total error would be the highest value of the individual prediction error given by (4):

$$\mathrm{PDF}_{\mathrm{total}} = \mathrm{PDF}_E(D_1) * \mathrm{PDF}_E(D_2) \tag{3}$$

$$\mathrm{PDF}_{\mathrm{total}} = \max(\mathrm{PDF}_E(D_1), \mathrm{PDF}_E(D_2)) \tag{4}$$

Another useful function window is used to extract temporal patterns from arriving series to deduce a complex event. The two basic window types are time window and tuple window. Time window depends on temporal connection between different events to evaluate a complex event, and the latter works on the number of defined events. The expression of overall prediction error for both types is the product of error for events covered by the window. If a CEP rule is defined as event D_1 is greater than T and event D_2 is greater than T for n successive readings, then the probability of total error is shown as:

$$\mathrm{PDF}_{\mathrm{total}} = n * \left(\mathrm{PDF}_E(D_1) * \mathrm{PDF}_E(D_2)\right) \tag{5}$$

In real-world scenarios, CEP rules tend to be more complex when combining the functionalities above but the total error can be calculated by replacing them with equations (2)-(5). We can derive the equation for overall prediction error by applying PDF equations once given the probability expression of prediction error for single data source. The approach to infer the equation for prediction error would be demonstrated at the end of results.

IV. ADWL BASED METHODLOGY

The main algorithm of proposed ADWL method is described in this section. There are mainly two steps included in the procedure. First the spectral analysis is exploited to find the desirable window size for training model. Then the prediction algorithm with dynamic prediction horizon is implemented to obtain the results.

A. Choosing Appropriate Training Window Size

The size of training window has considerable effects on the model training process and the accuracy of final prediction results. It is easy to understand that a small training window size would shorten the training time but introduced increased error or instability, while a large training window size can provide more precise results but add more complexity to the training model. Generally, the overall accuracy of prediction improves with the enlarging size of training data because large data set tends to cover all possible patterns required to be recognized. However, one problem here is that model after training is unable to trace irregular changes and leads to inaccurate readings if there are changes or modifications in the behavior of underlying data.

To overcome this problem and prevent the error accumulating and propagating to future predictions, researchers proposed to use method of dynamic window for model training. Before applying moving window on the machine learning model, there is a challenge to select the optimum window size. In this scenario, we decide to apply a generic method based on spectral analysis to find the proper value and validate our conclusion in later experiments. The key factor we exploit is the inherit periodicity of most real-world time series data. Here we consider an example of 24-hour temperature data to better explain this theory. The temperature data collected seems to be random and forms an irregular pattern on the monitor. But a repeating pattern could be observed among different figures formed by everyday temperature readings. Similar to the temperature example, in our traffic prediction research, road conditions vary every day while the peak periods and trough periods are not so stochastic from everyday traffic data. It is worth noting that our approach does not consider the collected data to be periodic but tries to seek for the highest periodic component in the pattern. In our method, the trained model is likely to learn all the local patterns and provides more satisfactory prediction results if the training window we find is exactly the same as the periodicity of underlying data.

Fast Fourier transform (FFT) might be the first algorithm in our mind for finding periodicity and is widely used in searching for the obvious peaks in the spectrogram of time series. However, one applicable requirement of FFT is the time series are equally spaced, which is not so practical for most real-world IoT data. In order to avoid the influence of missing values or

unexpected error, we exploit a method called least-square spectral analysis (LSSA) to find the highest periodic pattern in time series data. The method is also known as Lomb-Scargle method and was first proposed in [19].

Given a time series data x_i (i = 1, …, N) observed at times t_i respectively, and the mean and variance of time series are represented as $\bar{x}$ and σ^2. First for each angular frequency ω, the time-offset τ is computed by:

$$\tan(2\omega\tau) = \frac{\sum_{i=1}^{N} \sin(2\omega t_i)}{\sum_{i=1}^{N} \cos(2\omega t_i)} \tag{6}$$

then the LSSA periodogram is defined as:

$$P_N(\omega) = \frac{1}{2\sigma^2} \left\{ \frac{\left[\sum_{i=1}^{N} (x_i - \bar{x}) \cos \omega(t_i - \tau)\right]^2}{\sum_{i=1}^{N} \cos^2 \omega(t_i - \tau)} + \frac{\left[\sum_{i=1}^{N} (x_i - \bar{x}) \sin \omega(t_i - \tau)\right]^2}{\sum_{i=1}^{N} \sin^2 \omega(t_i - \tau)} \right\} \tag{7}$$

B. Prediction with Dynamic Prediction Window

The whole prediction algorithm could be divided into three consecutive parts. At first, a simple and common reinforcement learning method called Q-learning is applied to find a preliminary range for prediction window size. Then, the SVR method is called to train the machine learning model. Finally, the adaptive prediction algorithm with dynamic window is implemented to generate the results based on trained model and control the prediction error.

In the first part, we formulate the Q-learning process as follows. There are corresponding states and actions for targeted problem in Q-learning. For each step, the state is chosen from a set of states $S = \{s_1, s_2, \ldots, s_{ns}\}$ and action is selected from set of actions $A = \{a_1, a_2, \ldots, a_{na}\}$, where n_s and n_a represents the total number of states and actions respectively. After an action executed in a particular state, there would be feedback from the environment. The feedback is used to help the system learn about the outcome and is defined as reward r in our experiment, which is the absolute difference between the metric of our results p and the target value p_{tar}. This metric p is the combination of processing time and mean absolute percentage error (MAPE), which is an accurate metric for assessing the performance of prediction algorithms. MAPE here is represented as M and metric p are expressed as:

$$M = \frac{1}{N} \sum_{t=1}^{N} \left| \left(\frac{Y_a - Y_p}{Y_a} \right) \right| \times 100\% \tag{8}$$

$$p = 5M + \frac{t_p}{t_{\max}} \times 100\% \tag{9}$$

$$r = \left| p - p_{\text{tar}} \right| \tag{10}$$

where Y_a represents the actual value and Y_p stands for predicted value of data, t_p is the running time of prediction with a given window size, $t_{\max}$ represents the maximum processing time for all possible window values, and weight assigned to MAPE is five to highlight the importance of

accuracy. With the calculation of p, the next state for learning process could also be identified. Then, the Q value of current state and action is updated according to the observed new state s_{i+1} and reward value:

$$Q(s_i, a_j) \leftarrow (1-\alpha)Q(s_i, a_j) + \alpha[r + \gamma \min Q(s_{i+1}, a_j)] \tag{11}$$

where α stands for the learning rate and γ is the discount factor. The new Q value is determined by the current Q value of state-action pair selection, the reward received every step and the minimum Q value of observed next state.

In equation (11), the speed of learning process is determined by the learning rate α ($0 \leq \alpha \leq 1$) and discount factor γ ($0 \leq \gamma \leq 1$) decides the value or weight placed on the future reward. The whole learning process would depend more on immediate reward if γ is very small, while future and long-term reward would be the focus of learning if γ is set to a large number. When the current Q value is updated, a new action will be selected for the next state. Here we generate a random number u and compare it with parameter ε (usually a quite small number). If u is smaller than this parameter, an action will be chosen randomly from the action set A. On the contrary, action a_m with the minimum Q value for the next state will be our choice. This ε-greedy policy of Q-learning ensures that all the states and actions will be explored when implementing a large number of experiments. The process for selecting window size range is summarized in Algorithm 1.

Algorithm 1 Q-learning process for selection of initial window range

Initialization:

$s_i \in S$, $1 \leq i \leq n_s$, $a_j \in A$, $1 \leq j \leq n_a$, Q-table representation $Q(s_i, a_j)$

$\alpha = 0.5$, $\gamma = 0.9$, $\varepsilon = 0.01$

Iteration (for):

Generate a random number $u \in (0, 1)$

if $u < \varepsilon$

then select a_j randomly

else select action $a_j = a_m$ (a_m = argmin(Q-value))

end if

Implement SVR (Algorithm 4) with chosen action a_j

Obtain the result p and reward r

Observe and decide the next state s_{i+1} based on results

Update current Q-table with following:

$Q(s_i, a_j) \leftarrow (1-\alpha)Q(s_i, a_j) + \alpha[r + \gamma \min Q(s_{i+1}, a_m)]$

$s = s_{i+1}$

end

Return selected window range (W_1, W_2)

Initialization of Algorithm 4

As discussed previously in Ⅲ-C, the fundamental idea of SVM is to map the training data into higher dimensional space and find a hyperplane to maximize the margin in the feature space.

For regression problem in our example, if we consider a set of training data $\{(x_1, y_1), \ldots, (x_l, y_l)\}$, where x_i represents the input sample space and y_i denotes the corresponding value of each sample for $i = 1, \ldots, l$, where l is the training data size. Using Φ as the nonlinear transformation function, we form the function for SVR model as:

$$f(x) = (w \cdot \Phi(x)) + b \tag{12}$$

where $w \in R^n$, $b \in R$. The purpose of SVR is finding the proper value of w and b so that the cost function can be minimized. One of the most widely used function is the ε-intensive loss function [20] and is written as:

$$\Gamma[f(x) - y] = \begin{cases} |f(x) - y| - \varepsilon, for |f(x) - y| \geq \varepsilon \\ 0, otherwise \end{cases} \tag{13}$$

By minimizing cost function and solving related quadratic problem, we can complete the training of SVR model without knowing the mapping transformation. The procedure of finding SVR function is described in Algorithm 4 (see Appendix A) and more calculation details can be found in [21] and [22].

After obtaining the results from previous algorithms, our ADWL algorithm is applied to finish the predictions. To ensure a respectively high level of accuracy, we propose to apply an adaptive algorithm on the prediction horizon size. The fundamental thought of adaptive prediction window is to increase the prediction horizon size to provide more future predictions when the accuracy is high, and reduce the horizon size when the performance of our model decreases. Here, the prediction horizon size *win* is mainly controlled by the range returned in Algorithm 1. The value difference between actual data y_a and predicted data y_p is used as evaluation of model performance. The optimal range of evaluation parameter is defined as 5% to 10% initially and can be altered according to observations. More details and approach for adaptive prediction horizon are shown in Algorithm 2.

Algorithm 2 Prediction with dynamic horizon

Input: window size range (W_1, W_2) from Algorithm 1, prediction model $f(x)$

for data (x_a, y_a)
    Running Algorithm 4 (SVR)
    Predicting result $y_p = f(x_a)$
    $M = mean(abs((y_a - y_p)/y_a) * 100)$
    if $M > 10$, $W \leftarrow W - 1$
        if $W < W_1$, $W = W_1$
    else if $M < 5$, $W \leftarrow W + 1$
        if $W > W_2$, $W = W_2$
    end if
end
Return prediction horizon W

V. EXPERIMENTAL RESULTS

This section describes the data source we use, the implementation of proposed system model and the evaluation of prediction results. Then the performance of different algorithms are compared to find the optimum solution. Prediction error propagation scheme is studied and illustrated with expressions as well.

A. Data and Background Description

The first data set we use for experiments is provided by Madrid city online service [23]. The Madrid government has deployed thousands of road cameras and traffic sensors around the city for recording traffic features such as average speed and average traffic intensity. Traffic intensity refers to the number of passing vehicles per hour and is another important indicator of traffic states. With the information of real-time velocity and traffic intensity data, administrator in traffic center can have a clear vision about the current road conditions and take countermeasures to prevent traffic congestion. In our approach, historical data is utilized to find the optimum training window size and train the model, and real-time data is used to test the performance of trained model. The historical data is retrieved first which is available as a complete file. Real-time data in XML format is then accessed through public web service. The attributes of data set mainly consist of velocity, intensity, date and vehicle type.

To test the performance of our algorithm on different scenarios, other two data sets are chosen in our experiment. The second data set is an urban traffic speed data set which consists of 214 anonymous road segments [24]. The observations in this set were collected in August 2016 from Guangzhou, China. And the third data set contains traffic information from different locations in New York City, which is maintained by Traffic Management Center [25]. The data was collected in September 2018 and from major arterials and highways.

Besides, some parameters are defined for our experiments. We consider $\alpha = 0.5$, $\gamma = 0.9$, $\varepsilon = 0.01$, $p_{\text{tar}} = 65$ for Q-learning part, and C = 1000, $\varepsilon' = 0.1$ for prediction algorithm.

B. Evaluation

Before applying our prediction algorithm on obtained traffic condition (speed and intensity) data, we exploited spectral analysis method proposed earlier for calculating appropriate size of training window. The spectrogram for traffic speed data after applying LSSA is shown as Fig. 4. As the legend shows, x-axis refers to the frequency of data and y-axis indicates corresponding power spectral density. We can easily observe that data pattern reaches the highest point around 0.06, which is equivalent to 15-sample training window. Then we validated our analysis on the traffic data of March 2018 using several window sizes from 5 to 40. The reference we use here is mean absolute percentage error, which is an accurate metric for assessing the performance of prediction algorithms. MAPE for different training window is shown in the following table I and we conclude that error reaches the lowest level when training window size is 15 or multiple of 15 samples.

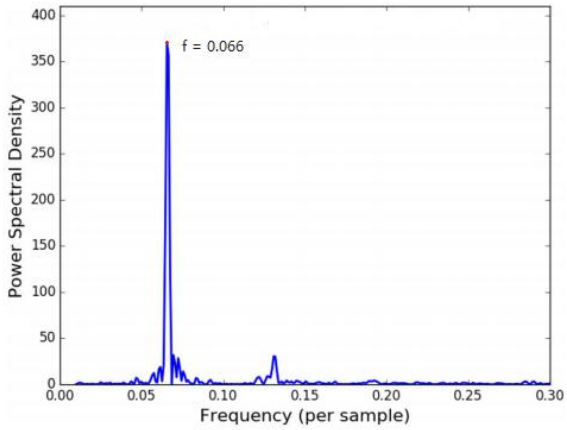


Fig. 4. Spectrogram for traffic speed data.

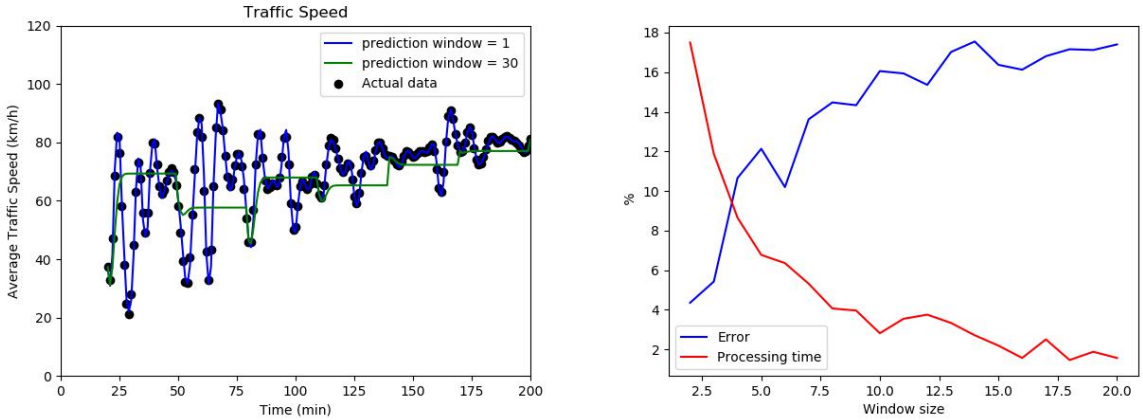
TABLE I

Error of one-month historical data with different training window size

Window Size	MAPE(%)
5	6.9219
10	6.8761
15	6.4265
20	6.6637
25	7.2970
30	6.4513
35	6.9017
40	7.3785

After determining the length of training window, we first applied SVR algorithm to the collected data to show the influence of prediction horizon. A large number and a small number is chosen as the value of prediction window size separately, and the performance for different choices are plotted together in Fig. 5(a). From the figure, we can observe that a smaller prediction window has much higher accuracy than larger window, while prediction horizon of 30 just captures the outline of varying data and loses lots of useful information.

However, we notice that running time of the whole algorithm is much longer when small number such as 1 is selected as prediction horizon. Taking both prediction error and processing time into consideration, the tendency of MAPE and algorithm running time with the increase of prediction window size is demonstrated in Fig. 5(b). As the figure shows, the error of prediction results rises quickly as prediction window becomes larger, in the meantime the processing time of prediction reduces rapidly. This result motivates us to find the optimum value or a proper range for prediction window, which could be summarized as trade-off between prediction accuracy and time cost of prediction.

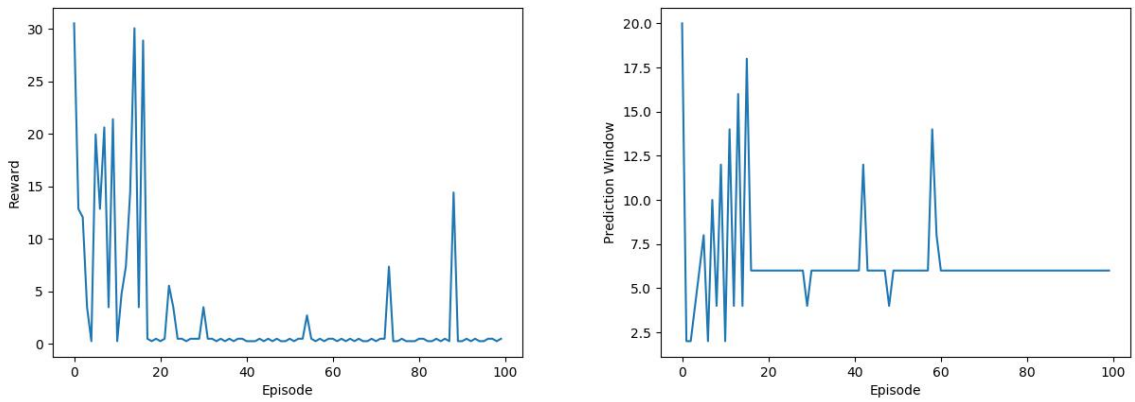


(a) (b)

Fig. 5. Effects of prediction window size.

(a) intuitive effects on prediction performance (b) effects on prediction error and prediction time.

Finding a better window range is also the main aim of applying Q-learning in the first part of our algorithm. We run the Q-learning algorithm as described in previous section and obtain the results as Fig. 6. The left figure shows the convergence of reward in Q-learning and the figure on the right shows the convergence of window size with the update of Q table. Each updating step is equal to one episode in the figure. According to Fig. 6, prediction window gets stabilized around six, so a desirable window size range could be determined based on this convergence value. It is possible to see some sudden fluctuations after getting convergence because of the policy used to select window size.



(a) (b)

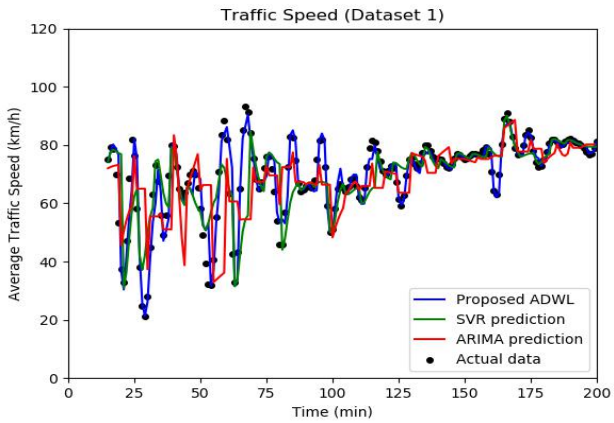
Fig. 6. Convergence results of (a) reward and (b) prediction window in Q-learning

After determining the range of prediction horizon, we picked four monitoring sites randomly from the data set to test our prediction model. The prediction tracks for collected traffic data from four locations are depicted in Fig. B1 and B2 (see Appendix B). As the figures show, the predicted values are closely tracking the trend of real data. The prediction error plays a vital role in achieving this because the error is integrated and the model is updated correspondingly to prevent the prediction error from propagating. The movement of prediction window size in Fig.

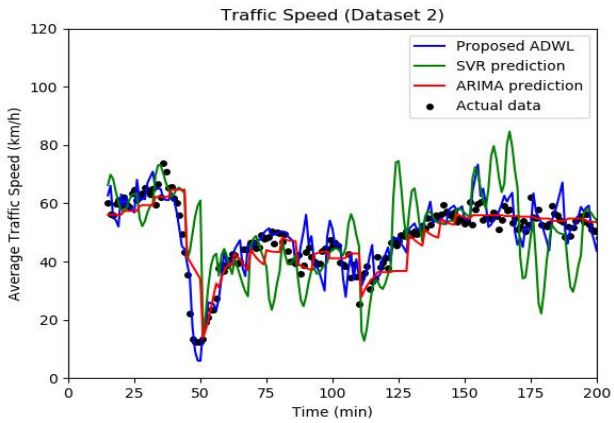
B3 indicates how prediction window adapts to the prediction error and guarantees the accuracy as well.

C. Model Comparison

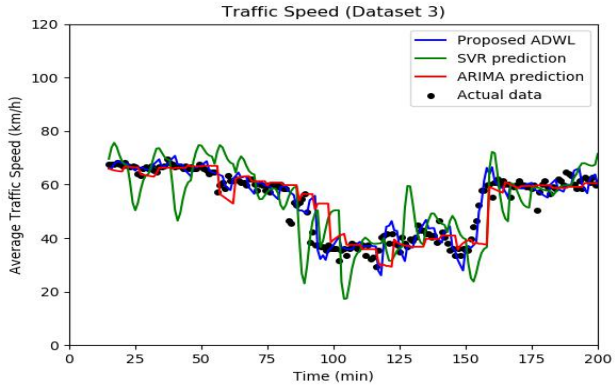
To better observe and evaluate the performance of proposed algorithm, we choose ARIMA model and original SVR algorithm as comparison. ARIMA is a widely used parametric model in traffic prediction cases and leads to relatively high accuracy in many researches [26]. SVR is a model-based algorithm and favored for predicting nonlinear traffic flow with both accuracy and simplicity [27]. we apply our proposed algorithm together with these two algorithms on three data sets to validate the applicability for different scenarios. Figure. 7 shows the performance of three algorithms on all selected data sets. It can be observed that proposed ADWL algorithm provides better prediction results with higher precision and stability. ARIMA model seems to predict the general trend but fails to track value changes, while SVR behaves too sensitive to the changes and generates more error.



(a)



(b)



(c)

Fig. 7. Prediction results of traffic speed data from (a) Madrid (b) Guangzhou and (c) New York data set.

In addition, we choose several commonly used regression models and implement them in python environment. The MAPE results for randomly picked locations are shown in Tables II and III. It can be concluded that proposed ADWL algorithm outperforms other regression algorithms with lower error and more reliable predictions. Compared to basic SVR algorithm, the improved algorithm decreases the prediction error by more than 40%. We notice that prediction accuracy of data in Location 2 is lower than the other three locations. The reason for such performance is that cumulative error for a certain period has relation with the uniformity of data set. If the data has a higher degree of dispersion than normal data samples, the precision of prediction results tends to be lower than expected. Calculation outcome demonstrates that the sample standard deviation from Location 2 is the highest among the deviation value from four selected locations, which leads to more error in predictions.

TABLE II

MAPE(%) for traffic speed data

Method	Loc 1	Loc 2	Loc 3	Loc 4
Proposed	6.45	11.61	6.45	4.19
SVM	11.85	19.58	14.83	11.56
LR	18.91	25.80	18.61	15.89
DT	12.44	19.82	14.58	11.42
RF	12.51	20.13	14.34	11.33

TABLE III

MAPE(%) for traffic intensity data

Method	Loc 1	Loc 2	Loc 3	Loc 4
Proposed	9.52	13.27	9.29	8.55
SVM	27.89	33.22	19.77	23.11
LR	34.54	33.16	32.52	31.45
DT	25.60	36.07	21.35	22.49
RF	25.84	35.83	21.37	22.47

As we mentioned in previous section, timeliness is also an important metric for measuring the value of prediction results. The prediction time is collected by recording the processing time for different algorithms in CPU. For better comparison, the prediction time from selected locations and the average performance for all the learning models are listed in Table IV. It can be concluded that conventional algorithms such as linear and decision tree regression spend relatively less time per prediction, however, prediction precision offered by these models is much lower than proposed ADWL model. Generally, time delay within one minute is accepted

as real-time for applications in ITS and it is reasonable to guarantee greater accuracy in road traffic prediction cases. As a result, the proposed algorithm shows better prediction performance with similar time cost compared with the other algorithms.

TABLE IV

Prediction time(s) on traffic speed data (per sample)

Method	Loc 1	Loc 2	Loc 3	Loc 4	Avg
Proposed	0.36	0.37	0.33	0.34	0.35
SVM	0.41	0.47	0.45	0.40	0.43
LR	0.31	0.33	0.30	0.31	0.31
DT	0.31	0.32	0.33	0.30	0.31
RF	0.32	0.35	0.32	0.30	0.32

（LR: Linear Regression, DT: Decision Trees, RF: Random Forest）

In our experiments, we spotted a congestion point in the training data set 1 and further validated the adaptability of proposed algorithm with this point. The figure below shows the difference between performance of conventional SVR and our improved algorithm for predicting. The congestion point arises around 175th min in Fig. 8 when detected speed drops to zero immediately. The adaptive algorithm we use tracks the actual trend of data perfectly and captures the sudden congestion while conventional SVR algorithm neglects this congestion event.

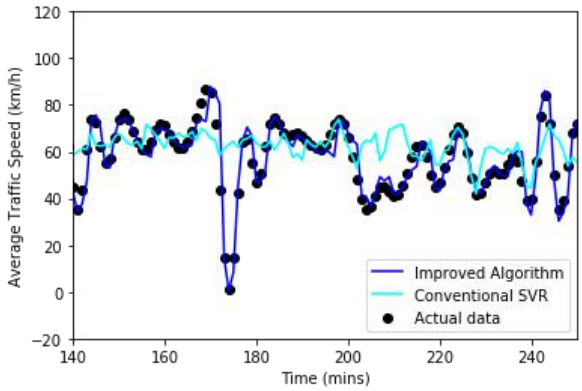


Fig. 8. Performance at congestion point

In order to understand how error model works, a simple rule for generating events like bad traffic or congestion is shown first. Algorithm 3 describes the rule combining traffic speed and traffic intensity in detail, where V stands for the speed, I stands for the traffic intensity, and T represents the predefined threshold value. When the average vehicle speed and average traffic intensity is less than defined threshold value for three consecutive predictions, CEP would generate a complex event called congestion. In this way, the predicted upcoming events are presented to traffic management center and precautions can be made in advance. This is an example of how CEP generates complex events and more complex rules may be associated with

weather conditions [28] and tweet media data [29].

Algorithm 3 CEP Rule Example

for $(V, I) \in Window(3)$
  if ($V_t < T_{\text{speed}}$ and $I_t < T_{\text{intensity}}$ &
    $V_{t+1} < T_{\text{speed}}$ and $I_{t+1} < T_{\text{intensity}}$ &
    $V_{t+2} < T_{\text{speed}}$ and $I_{t+2} < T_{\text{intensity}}$)
  then Generate complex event Congestion
  end if
end

The total error propagated to the event generation end is determined by the relevant CEP rule. For the rule mentioned in Algorithm 3, the expression for total error is derived as:

$$\text{PDF}_{\text{total}} = n * \left(\text{PDF}_E(V) * \text{PDF}_E(I)\right) \tag{14}$$

where n is the window size, and other two multiplier terms represent probability density function of speed error and intensity error. Moreover, we can get the expression for the overall error if given respective expressions for $\text{PDF}_E(V)$ and $\text{PDF}_E(I)$.

In order to obtain the probability density functions of traffic speed and traffic intensity, the prediction error is calculated based on one-month historical data. Then the result is plotted in an histogram which provides us with better observation about probability distribution of prediction error. We choose data from Location 1 as example and its error distribution is shown in Fig. 9. Normally, Gaussian distribution is chosen for error propagation modeling in statistics due to its applicability for various models in real world. However, the modeling performance of Gaussian distribution is unconvincing as it fails to fit the deviated points on the error histogram in our experiment.

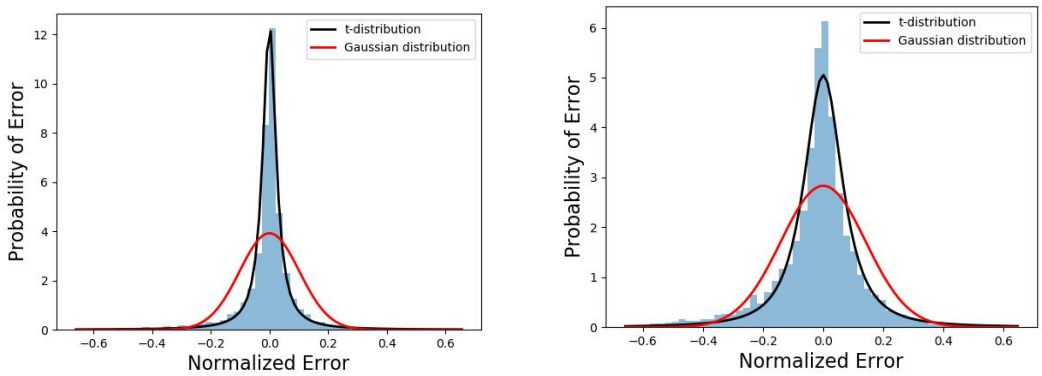


Fig. 9. Error distribution for traffic speed (a) and intensity (b) in Location 1.

As a better model is needed to fit the histogram, we turn our attention to applying student's t distribution for error modeling [30]. The student's t distribution has the nature of heavy tail and light top, which is the reason why it models the outliers better. Besides, t distribution may exploit its degree of freedom to change the shape of distribution with the propagating error. The

probability density function of Gaussian and *t* distribution are shown as (15)-(16):

$$\mathrm{PDF}_E(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2} \tag{15}$$

$$\mathrm{PDF}_E(x) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\pi\nu}\sigma\Gamma(\frac{\nu}{2})}\left[1+\frac{1}{\nu}(\frac{x-\mu}{\sigma})^2\right]^{-\frac{\nu+1}{2}} \tag{16}$$

where μ represents the mean and σ represents the standard deviation for both distributions, and v refers to the degree of freedom parameter for student's *t* distribution. The variance and leptokurtic property can be adjusted with the parameter v. For example, if we decrease the value of v, the tails of the distribution will become higher. The curve in Fig. 9 proves *t* distribution has better fitting performance than Gaussian distribution and is a desirable choice for error modeling.

VI. CONCLUSION

Online open data and applications are potential sources to be used to gather traffic flow information. In this paper, we proposed and implemented a CEP-ML combined system for predicting complex traffic events with the help of open source tools. In our system, ML component is used to learn historical data and CEP component provides the solution for processing real-time data. To achieve better performance in machine learning process, we adopt an adaptive algorithm for near real-time prediction. The method proposed in the report exploits optimum training window and dynamic prediction horizon therefore preventing error propagation. We also use Q-learning method to help determine the range for horizon changing. The verification experiments on different scenarios show that our prediction algorithm provides better prediction results than common ML algorithms and achieves an accuracy of over 95% on real-world traffic data. Furthermore, the functionality of early prediction may assist traffic administrators to take measures ahead of time, which is required for future ITS.

VII. FUTURE WORK

As for future research, more elements are supposed to be considered to improve the overall performance of system model. Weather conditions, social media and nearby activities such as sports games are expected to be incorporated into the prediction model to closely fit the practical situations. The system model is generic and can be utilized for other applications besides ITS as we discussed, but the selection of window size and scope of real-time are dependent on the application scenarios. Therefore, testing the performance of proposed model on different prediction problems such as health monitoring and goods supply chain would be the next step.

Another direction is the large-scale implementation of the whole architecture. The ML component in our model can support relatively large data sets but still has limitation on the amount of processing data. Besides, the energy cost of system model can be a tricky issue when dealing with a massive influx of data. To make our system scalable and applicable for high-intensity analytic, we would turn to Apache Spark because of its access to various machine

learning algorithms and resilient distributed data sets [31]. More researches would be completed to evaluate its capability for predicting complex events.

REFERENCES

[1] G. A. Akpakwu, B. J. Silva, G. P. Hancke and A. M. Abu-Mahfouz, "A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges," in *IEEE Access*, vol. 6, pp. 3619-3647, 2018.

[2] Y. Lv, Y. Duan, W. Kang, Z. Li, and F. Y. Wang, "Traffic flow prediction with big data: A deep learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 865 – 873, April 2015.

[3] L. Li, X. Su, Y. Zhang, Y. Lin, and Z. Li, "Trend modeling for traffic time series analysis: An integrated study," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 6, pp. 3430 – 3439, Dec 2015.

[4] G. Cugola and A. Margara, "Processing flows of information: From data stream to complex event processing," *ACM Comput. Surveys*, vol. 44, no. 3, pp. 1 – 62, Jun. 2012.

[5] C.-W. Tsai, C.-F. Lai, M.-C. Chiang, and L. T. Yang, "Data mining for Internet of Things: A survey," *IEEE Commun. Surveys Tuts*., vol. 16, no. 1, pp. 77 – 97, 1st Quart., 2014.

[6] L. J. Fülöp et al., "Predictive complex event processing: A conceptual framework for combining complex event processing and predictive analytics," in *Proc. 5th Balkan Conf. Inform*., 2012, pp. 26 – 31.

[7] Y. Wang and K. Cao, "A proactive complex event processing method for large-scale transportation Internet of Things," *Int. J. Distrib. Sensor Netw*., vol. 10, no. 3, 2014, Art. no. 159052.

[8] A. Valsamis, K. Tserpes, D. Zissis, D. Anagnostopoulos, and T. Varvarigou, "Employing traditional machine learning algorithms for big data streams analysis: The case of object trajectory prediction," *J. Syst. Softw.*, vol. 127, pp. 249 – 257, 2017.

[9] B. Thomas, F. Jose, S. Jordi, A. Almudena, and T. Wolfgang, "Real time traffic forecast," AtoS Sci. White Paper, vol. 2013, 2013.

[10] P. S. Kumar and S. Pranavi, "Performance analysis of machine learning algorithms on diabetes dataset using big data analytics," *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS)*, Dubai, 2017, pp. 508-513.

[11] S. Ren, L. Han, Z. Li and B. Veeravalli, "Spatial-temporal traffic speed bands data analysis and prediction," *2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, Singapore, 2017, pp. 808-812.

[12] Yuan-yuan Chen, Y. Lv, Z. Li and F. Y. Wang, "Long short-term memory model for traffic congestion prediction with online open data," *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Rio de Janeiro, 2016, pp. 132-137.

[13] D. Wang, J. Xiong, Z. Xiao and X. Li, "Short-Term Traffic Flow Prediction Based on Ensemble Real-Time Sequential Extreme Learning Machine Under Non-Stationary Condition," *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, Nanjing, 2016, pp. 1-5.

[14] Williams, B.M., Hoel, L.A.: "Modeling and forecasting vehicular traffic flow as a seasonal ARIMA process: theoretical basis and empirical results", *J. Trans. Eng*., 2003, 129, (6), pp. 664 – 672.

[15] S. V. Kumar and L. Vanajakshi, "Short-term traffic flow prediction using seasonal ARIMA model with limited input data," *Eur. Transp. Res. Rev.*, vol. 7, p. 21, Sep. 2015.

[16] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Stat. Comput.*, vol. 14, no. 3, pp. 199

– 222, 2004.

[17] M. Deshpande and P. R. Bajaj, "Performance analysis of support vector machine for traffic flow prediction," *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, Jalgaon, 2016, pp. 126-129.

[18] A. Akbar, F. Carrez, K. Moessner and A. Zoha, "Predicting complex events for pro-active IoT applications," *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Milan, 2015, pp. 327-332.

[19] W. H. Press and G. B. Rybicki, "Fast algorithm for spectral analysis of unevenly sampled data," *Astrophys. J.*, vol. 338, pp. 277-280, Mar. 1989.

[20] B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds, "Using support vector support machines for time series prediction," in *Advances in Kernel Methods*, Cambridge, MA: MIT Press, 1999, pp. 242 – 253.

[21] Chun-Hsin Wu, Jan-Ming Ho and D. T. Lee, "Travel-time prediction with support vector regression," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, no. 4, pp. 276-281, Dec. 2004.

[22] D. Wu, "Time Series Prediction for Machining Errors Using Support Vector Regression," *2008 First International Conference on Intelligent Networks and Intelligent Systems*, Wuhan, 2008, pp. 27-30.

[23] Traffic data of city of Madrid. [Online]. Available: http://informo.munimadrid.es/informo/tmadrid/pm.xml.

[24] Urban Traffic Speed Dataset of Guangzhou. [Online]. Available: https://github.com/sysuits/urban-traffic-speed-dataset-Guangzhou#urban-traffic-speed-dataset-of-guangzhou

[25] NYC Real-Time Traffic Speed Data. [Online]. Available: https://data.cityofnewyork.us/Transportation/Real-Time-Traffic-Speed-Data/qkm5-nuaq

[26] I. Kalamaras et al., "An Interactive Visual Analytics Platform for Smart Intelligent Transportation Systems Management," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 487-496, Feb. 2018.

[27] W. Hu, L. Yan, K. Liu and H. Wang, "PSO-SVR: A Hybrid Short-term Traffic Flow Forecasting Method," *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, Melbourne, VIC, 2015, pp. 553-561.

[28] D. Zhang and M. R. Kabuka, "Combining Weather Condition Data to Predict Traffic Flow: A GRU Based Deep Learning Approach," *2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress*, Orlando, FL, 2017, pp. 1216-1219.

[29] N. Bichu and A. Panangadan, "Analyzing social media communications for correlation with freeway vehicular traffic," *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation*, San Francisco, CA, 2017, pp. 1-7.

[30] W. Rafique, S. M. Naqvi, P. J. B. Jackson, and J. A. Chambers, "IVA algorithms using a multivariate Student's t source prior for speech source separation in real room environments," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Brisbane, QLD, Australia, Apr. 2015, pp. 474 – 478.

[31] COSMOS. (2016). D.4.1.3. *Information and Data Lifecycle Management: Design and Open Specification (Final)*. Accessed on Apr. 6, 2017.

APPENDIX A

The explanation and basic implementation of support vector regression (SVR) algorithm is included in this part.

Algorithm 4 SVR prediction algorithm

Input: training data set $\{(x_1, y_1), \ldots, (x_l, y_l)\}$, $x_i \in \mathrm{R}^n$, $y_i \in \mathrm{R}$ ($i = 1, 2, \ldots, l$)

Output: prediction model $f(x)$

Initialization:

$w \in \mathrm{R}^n$, $b \in \mathrm{R}$, $C = 10^3$, $\varepsilon = 0.01$, $\gamma = 0.1$

Lagrange multipliers $0 < \alpha_i, \alpha_i^* \leq C$, $\eta_i, \eta_i^* > 0$

Iteration:

$\mathrm{f}(x) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) K(x_i, x) + b$, K—RBF kernel function

Minimize cost: $|f(x) - y| - \varepsilon$

Dual problem: $\min \frac{1}{2} \sum_{i=1}^{l} \left(\alpha_i - \alpha_i^*\right)^T K\left(x_i, x_j\right)\left(\alpha_i - \alpha_i^*\right) + \varepsilon \sum_{i=1}^{l} \left(\alpha_i + \alpha_i^*\right) + \sum_{i=1}^{l} y_i \left(\alpha_i - \alpha_i^*\right)$

for $i = 1, \ldots, l$

$$\min\left[\frac{1}{2}\left(\alpha_i - \alpha_i^*\right)^T K\left(x_i, x_j\right)\left(\alpha_i - \alpha_i^*\right) + \alpha_i (y_i + \varepsilon) - \alpha_i^* \left(y_i - \varepsilon\right)\right]$$

$$\left(\alpha_i - \alpha_i^*\right) = \operatorname{argmin}\left[\frac{1}{2}\left(\alpha_i - \alpha_i^*\right)^T K\left(x_i, x_j\right)\left(\alpha_i - \alpha_i^*\right) + \alpha_i (y_i + \varepsilon) - \alpha_i^* \left(y_i - \varepsilon\right)\right]$$

$w = \alpha^* - \alpha$

$b = \mathrm{mean}[y_i - \varepsilon - \Sigma (\alpha_i - \alpha_i^*) K(x_i, x)]$

end

APPENDIX B

This appendix contains evaluation results of proposed algorithm for four different locations in data set 1 (Madrid city service) and variety of prediction horizon.

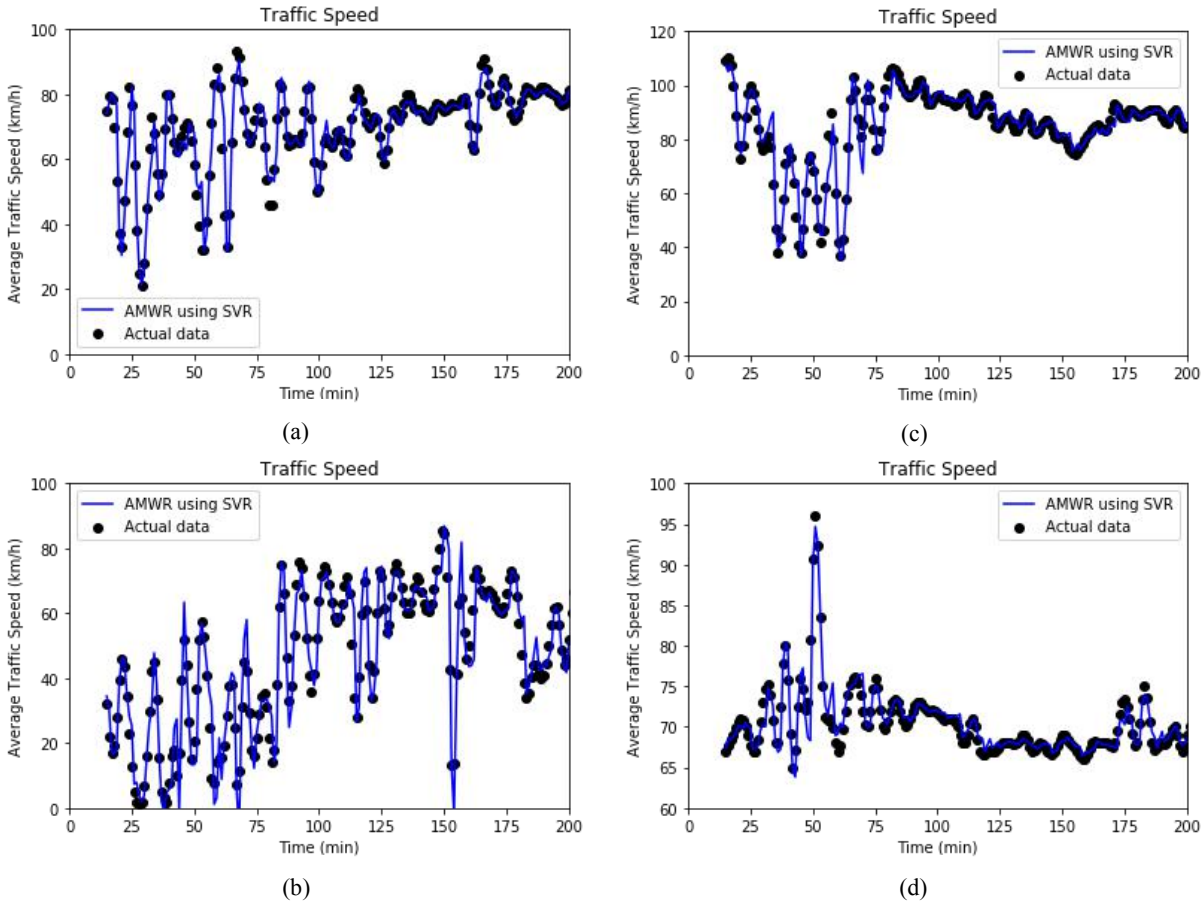


Fig. B1. Prediction results on average traffic speed data from four locations 1-4 (a-d).

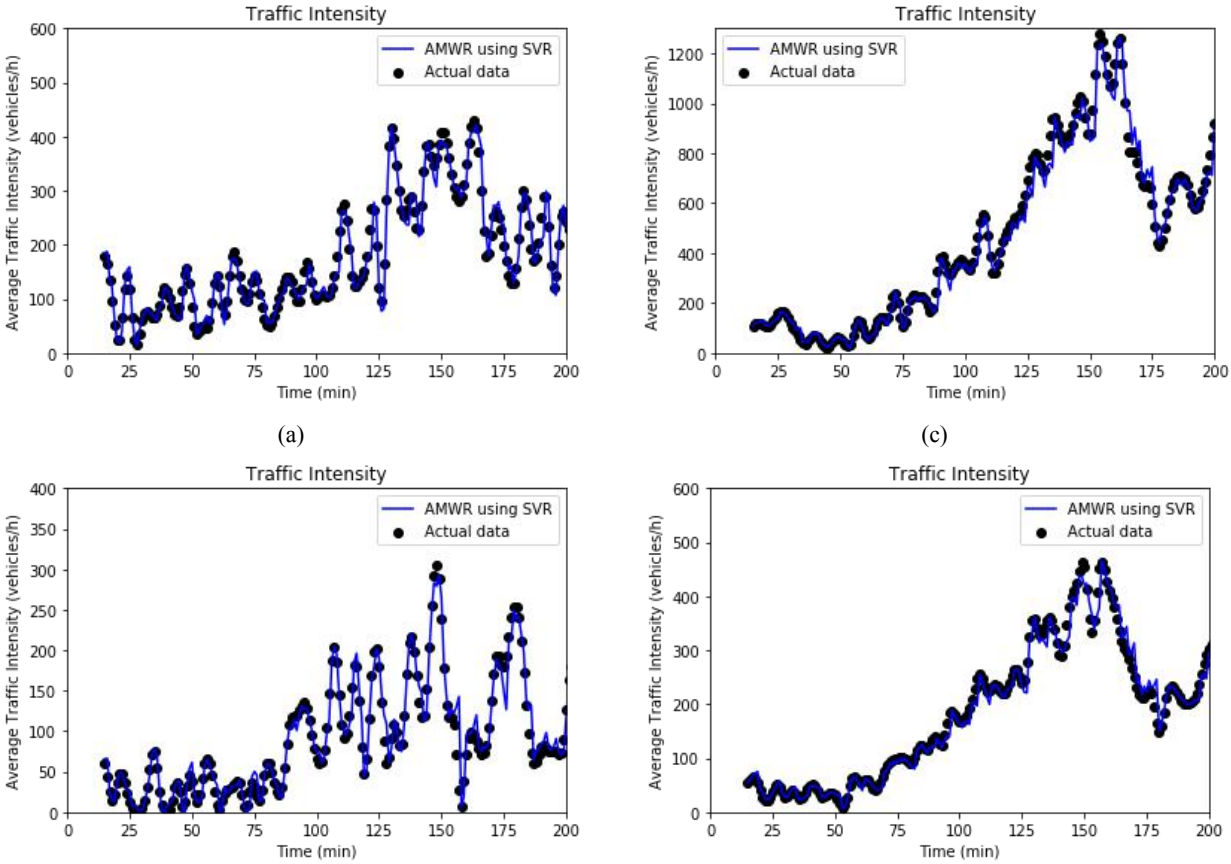


Fig. B2. Prediction results on average traffic intensity data from four locations 1-4 (a-d).

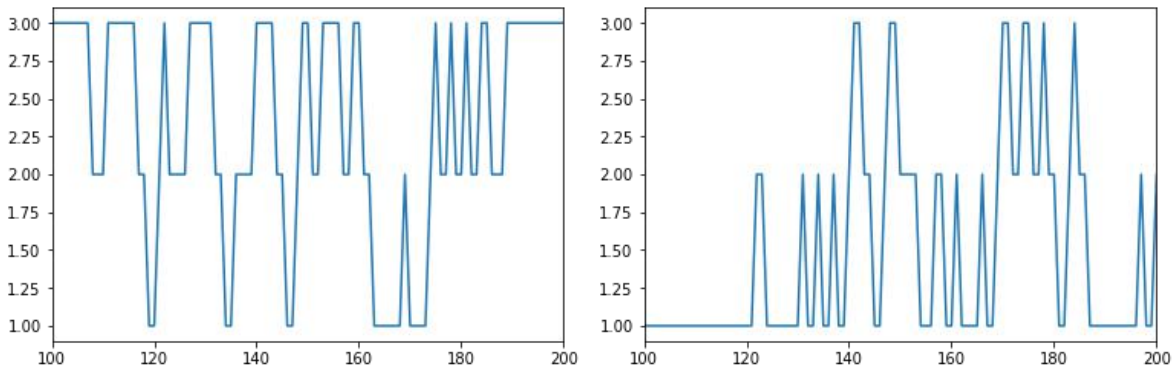


Fig. B3. Changing tendency of prediction horizon for traffic speed and traffic intensity

APPENDIX C

Some codes for implementing machine learning algorithms and calculating prediction error in Python environment are listed here.

a. Python Code for Machine Learning on Historical Data

```
'''this files runs algorithm on historical data, need to change the path of main file in input function'''

import numpy as np
import csv
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import SVR
from scipy.stats import norm, expon    , ks_2samp, kstest, t
from pandas import *
from sklearn.neighbors import KernelDensity
from sklearn.tree import DecisionTreeRegressor
#from sklearn.utils import check_arrays
TrainingWindow = 15

#pre-processing function
def pre_processing(df):
    #code for extracting data for particular time
    hour = df.index.hour
    selector_new = (( 1 <= hour) & (hour <= 23))
    df1 = df[selector_new]
    #resampling in order to uniform the time series
    df2 = df1.resample('5min')
    #interpolation to fill missing values
    df3 = df2.interpolate(method ='cubic')
    return df3

#function to model the prediction error
def error_model(data):
    #fitting a normal distribution
    #fitting t dist
    df, mu, std = t.fit(data, floc = 0)
```

```
    #fitting norm dist
    mu1, std1 = norm.fit(data, floc=0)
    print (' T dist parameteres are', a, mu, std)
    print ('Gaussian parameters are', mu1, std1)

    plt.figure()
    # Plot the histogram.
    plt.hist(data, bins=25, normed=True, alpha=0.5)
    # Plot the PDF.
    xmin, xmax = plt.xlim()
    x = np.linspace(xmin, xmax, 100)
    p = t.pdf(x, df, loc = mu, scale = std)
    p1 = norm.pdf(x, loc = mu1, scale = std1)
    kde_data = data[:, np.newaxis]
    X_plot = np.linspace(xmin, xmax, 1000)[:, np.newaxis]
    # Gaussian KDE
    kde = KernelDensity(kernel='gaussian', bandwidth=0.035).fit(kde_data)
    log_dens = kde.score_samples(X_plot)
    #plt.plot(X_plot[:, 0], np.exp(log_dens), linewidth = 2)
    plt.plot(x, p, 'k', linewidth=2, label = 't-distribution')
    plt.plot(x, p1, 'r', linewidth=2, label = 'Gaussian distribution')
    title = "Fit results: mu = %.2f,    std = %.2f" % (mu, std)
    #plt.title(title)
    plt.xlabel('Normalized Error', fontsize = 16)
    plt.ylabel('Probability of Error', fontsize = 16)
    #plt.ylim(0,6)
    #plt.title('Sensor 1', fontsize = 17)
    plt.legend()
    plt.show()

#main function
if __name__ == "__main__":
    #define window sizes
    #TrainingWindow = 15
    #creating DataFrame object
    df = pd.DataFrame()
    #reading csv file into DataFrame object
    dfa = pd.read_csv('C:/Users/THINKPAD/Desktop/M.Eng Pro/PredictiveAnalytics/03-2018.csv',
parse_dates=['fecha'], index_col='fecha', sep = ';')
    #location selection: PM10005, PM10344, 18RV21PM01 and 03FL20PM01/ 90EL69PM01 42001
14XC71PM01
```

```
dfa = dfa[dfa['identif'] == '14XC71PM01']
df_intensidad_1 = dfa[['intensidad']]
#'vmed' = velocity, 'intensidad' = 'intensity'.
#calling pre-processing function
df_total = pre_processing(df_intensidad_1)
#model implementation
Y1= df_total.values
std_Y1 = np.asarray(Y1)
#print "standard deviation of Input is", std_Y1.std()

#regression method in the library needs array in specific format.
#Converting input and output array in the required format
X=[]
Y=[]
Y11 = []

for i in range(len(df_total)):
    Y11.append(abs(Y1[i][0]))

#getting rid of 0`s    (1~121 for speed, 1~1301 for intensity)
for i in range(len(Y11)):
    if Y11[i] > 1 and Y11[i] < 1301:
        a=Y11[i]
        Y.append(a)
for i in range(len(Y)):
    X.append([i])

#initializes the model
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
X_predicted = []
Y_predicted = []

#setting total length to iterate. as i is iterated through total length and
#when we use (i+24) it should be in data
total = len(X)-TrainingWindow -2
#iterations of i for every n, where n is window size. 3 in this case
PredictionWindowArray =[]
MAPE = []
TrueError = []
PredictionWindow = 2
i=0
```

```
while(i <= total):
    X_training = X[i:i+TrainingWindow]
    Y_training = Y[i:i+TrainingWindow]
    X_testing = X[i+TrainingWindow:i+TrainingWindow+PredictionWindow]
    y_testing = Y[i+TrainingWindow:i+TrainingWindow+PredictionWindow]
    i = i + PredictionWindow

    #training and prediction
    y_rbf = svr_rbf.fit(X_training, Y_training).predict(X_testing)
    #a=y_rbf[0]
    #for j in y_rbf[0]:
    #Y_predicted.append(a)
    for j in y_rbf:
        Y_predicted.append(j)
    for k in X_testing:
        X_predicted.append(k)

    #calculating error (y_true - y_pred)/y_true
    TempError = []
    for m in range(len(y_testing)):
        a = abs((y_testing[m] - y_rbf[m])/y_testing[m])
        if a < 0.60:
            c = a
        b= ((y_testing[m] - y_rbf[m])/y_testing[m])
        if abs(b) < 0.60:
            d = b
        TempError.append(c)
        TrueError.append(d)

#converting in an array
    TempError1 = np.asarray(TempError)
#taking mean
    TempMAPE = (TempError1.mean())*100
    MAPE.append((TempMAPE))
    #it is predicting every value in a loop. all predicted values are appended in
    #an array for overall error and graph
    #if error is greater then 10%, decrease the prediction window size
    if TempMAPE > 10:
        PredictionWindow = PredictionWindow -1
        if PredictionWindow == 0:
```

```
                    PredictionWindow = 1
                PredictionWindowArray.append(PredictionWindow)
            #if error is less then 5%,increase the prediction window size
            if TempMAPE < 5:
                PredictionWindow = PredictionWindow +1
                if PredictionWindow > 3:
                    PredictionWindow = PredictionWindow -1
                PredictionWindowArray.append(PredictionWindow)
            else:
                PredictionWindowArray.append(PredictionWindow)

        MAPEArray = np.asarray(MAPE)
        print ('error is', MAPEArray.mean())

        #Extracting data for testing and plotting
        Y_testing = Y[TrainingWindow:]
        X_testing = X[TrainingWindow:]
        #plot original/actual data
        plt.scatter(X_testing, Y_testing, c='k', label='Actual data')
        #plot predicted data
        plt.plot(X_predicted, Y_predicted, c='b', label = 'AMWR using SVR')
        #plt.scatter(X_predicted, Y_predicted, c='red', label = 'Predicted data')
        print ('the length of testing and prediction array is', len(Y_testing), len(Y_predicted))
        x_step =[]
        for i in range(len(PredictionWindowArray)):
            x_step.append(i)

        plt.xlim(0,200)
        plt.ylim(0,1200)          # may change for different locations
#       plt.xlabel('Time (min)')
#       plt.ylabel('Average Traffic Speed (km/h)')
#       plt.title('Traffic Speed')
        plt.xlabel('Time (min)')
        plt.ylabel('Average Traffic Intensity (vehicles/h)')
        plt.title('Traffic Intensity')
        plt.legend()

        plt.figure(2)
        plt.plot(PredictionWindowArray)
        plt.xlim(100,200)
        #converting in an array
```

```
    TrueError1 = np.asarray(TrueError)
    data = TrueError1
    #calling error modelling function
    error_model(TrueError1)
    plt.show()
```

b. Python Code for Machine Learning on Real-time Data

"'this program reads online data from Madrid website, preprocess it, and applies proposed algorithm to predict next three readings'"

```
import http
from urllib.request import urlopen
from urllib.request import Request
import xml.etree.ElementTree as ET
import urllib
import re
import io, random
import pandas as pd
import time
from datetime import datetime
from sklearn.svm import SVR
import datetime
from lxml import etree

#constant parameters
TrainingWindow = 15       #window size is found from historical data
PredictionWindow = 3      #prediction window/prediction horizon
time_sampling = 300       #Should be equal to data refreshing time
#function to read data
def data_traffic_read():
    req = Request(url='http://informo.munimadrid.es/informo/tmadrid/pm.xml')
    f = urlopen(req)
    try:
        xml_str = f.read()
    except http.client.IncompleteRead as e:
        xml_str = e.partial
    parser = etree.XMLParser(recover=True)
    root = ET.fromstring(xml_str, parser=parser)
    list = []
    for location in root.findall('pm'):
        codigo = location.find('idelem').text
```

```
            #here 'if' statement can be added to look for IDs with PM
            #and publish it into different topics
            flag = re.match('pm',codigo)
            if (flag):
                codigo = location.find('idelem').text
                intensity = float(location.find('intensidad').text)
                speed = float(location.find('velocidad').text)
                occupancy = float(location.find('ocupacion').text)
                date = (datetime.datetime.utcnow())
                error = location.find('error').text
                if error == 'N':
                    message = {'Id':codigo, 'TrafficIntensity':intensity, 'TrafficSpeed': speed,
'TrafficOccupancy':occupancy, 'Date_UTC': date}
                    list.append(message)
        return list

#function to apply prediction based on svr with rbf kernel
def pred(df):
        X = []
        Y = []
        for index, row in df.iterrows():
            X.append([index.hour, index.minute])
            Y.append(row)
        #initializes the model
        svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
        y_rbf = svr_rbf.fit(X, Y.values.ravel())

        #predicting for next readings
        #first we take the time for the latest reading
        length = len(df)
        df_pred = df[length-1:length]
        for index, row in df_pred.iterrows():
            time_last = index
        print ("time_last is {}".format(time_last))
        #extracting time for next 3 predictions
        X_pred = []
        for i in range(3):
            time_new = time_last + datetime.timedelta(seconds = (i+1)*time_sampling)
            X_pred.append([time_new.hour, time_new.minute])
        Y_pred = y_rbf.predict(X_pred)
        return X_pred, Y_pred
```

```
#adaptive movie window regression function
def AMWR(df):
    #extracting last readings equivalent to window size
    length = len(df)
    df1 = df[length-TrainingWindow:length]
    df_speed = df1[['Date_UTC','TrafficSpeed']]
    df_speed1 = df_speed.set_index('Date_UTC')
    X_speed, Y_speed = pred(df_speed1)
    df_intensity = df1[['Date_UTC','TrafficIntensity']]
    df_intensity1 = df_intensity.set_index('Date_UTC')
    X_intensity, Y_intensity = pred(df_intensity1)
    for i in range(3):
        print ("Expected traffic speed at {}:{} is {}".format(X_speed[i][0], X_speed[i][1], int(Y_speed[i])))
        print ("Expected traffic intensity at {}:{} is {}".format(X_intensity[i][0], X_intensity[i][1],
int(Y_intensity[i])))

#main function
if __name__ == '__main__':
    total_list = []
    while(1):
        list = data_traffic_read()
        total_list = total_list + list
        df = pd.DataFrame(total_list)
        #print df
        print (df['Id'] == '6762')
        df1 = df[df['Id'] == '6762']
        #print len(df1)
        if len(df1) >= TrainingWindow:
            print ("calling prediction algorithm")
            print ("df1 is ", df1)
            AMWR(df1)
        else:
            pass
        print ("i am sleeping")
        time.sleep(time_sampling)
```

c. Python Code for Congestion Detection

"'this program compares the performance of proposed method with simple regression model when predicting a congestion point'"

```
import numpy as np
import csv
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import SVR
from scipy.stats import norm, expon    , ks_2samp, kstest, t
from pandas import *
from sklearn.neighbors import KernelDensity
from sklearn.tree import DecisionTreeRegressor

def model():
    #reading csv file into DataFrame object
    dfa = pd.read_csv('C:/Users/THINKPAD/Desktop/M.Eng Pro/PredictiveAnalytics/03-2018.csv',
parse_dates=['fecha'], index_col='fecha', sep = ';')
    dfa = dfa[dfa['identif'] == '18RV21PM01']
    df_intensidad_1 = dfa[['vmed']]
    #code for extracting data for particular time
    hour = df_intensidad_1.index.hour
    selector_new = (( 1 <= hour) & (hour <= 23))
    df_total_1 = df_intensidad_1[selector_new]
    #interpolation using cubic method for missing values
    df_total_1 = df_total_1.resample('5min')
    df_total_1 = df_total_1.interpolate(method ='cubic')
    hours = df_total_1.index.hour
    mins = df_total_1.index.minute
    df_total = df_total_1
    Y1= df_total.values
    std_Y1 = np.asarray(Y1)
    print ("standard deviation of Input is", std_Y1.std())

    #regression method in the library needs array in specific format.
    #Converting input and output array in the required format
    X=[]
    Y=[]
    Y11 = []
    for i in range(len(df_total)):
        Y11.append(abs(Y1[i][0]))
```

```
    #getting rid of 0`s and outliers
    for i in range(len(Y11)):
        a=Y11[i]+1
        Y.append(a)
    for i in range(len(hours)):
        X.append([hours[i], mins[i]])
    I = []
    for i in range(len(hours)):
        I.append([i])
    X_predicted = []
    Y_predicted = []
    X_training = X[:1000]
    Y_training = Y[:1000]

    #initializes the model
    svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
    y_rbf = svr_rbf.fit(X_training, Y_training)
    TotalError = []
    X_testing = X[50:1300]
    Y_testing = Y[50:1300]
    index = I[50:1300]
    a = y_rbf.predict(X_testing)
    return index, a

#define window sizes
TrainingWindow = 20
#creating DataFrame object
df = pd.DataFrame()
#reading csv file into DataFrame object
dfa = pd.read_csv('C:/Users/THINKPAD/Desktop/M.Eng Pro/PredictiveAnalytics/03-2018.csv',
parse_dates=['fecha'], index_col='fecha', sep = ';')
dfa = dfa[dfa['identif'] == '18RV21PM01']
#feature. intensidad or vmed
df_intensidad_1 = dfa[['vmed']]

#code for extracting data for particular time
hour = df_intensidad_1.index.hour
print (len(hour))
selector_new = (( 1 <= hour) & (hour <= 23))
df_total_1 = df_intensidad_1[selector_new]
#resampling. Original data has 15 mint sampling period
```

```
df_total_1 = df_total_1.resample('5min')

#interpolation using cubic method
df_total_1 = df_total_1.interpolate(method ='cubic')
df_total = df_total_1
hours = df_total.index.hour
mins = df_total.index.minute
#df_total = concat([df_total_1])
Y1= df_total.values
std_Y1 = np.asarray(Y1)
print ("standard deviation of Input is", std_Y1.std())

#regression method in the library needs array in specific format.
#Converting input and output array in the required format
X=[]
Y=[]
Y11 = []
for i in range(len(df_total)):
    Y11.append(abs(Y1[i][0]))
for i in range(len(Y11)):
    a=Y11[i]+1
    Y.append(a)
for i in range(len(Y)):
    X.append([i])
features = []
for i in range(len(hours)):
    features.append([hours[i], mins[i]])
print (len(X), len(Y))

#initializes the model
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
X_predicted = []
Y_predicted = []
total = len(X)-TrainingWindow -2

#iterations of i for every n, where n is window size. 3 in this case
PredictionWindowArray =[]
MAPE = []
TrueError = []
PredictionWindow = 2
i=0
```

```
while(i <= total):
    X_training = X[i:i+TrainingWindow]
    Y_training = Y[i:i+TrainingWindow]
    X_testing = X[i+TrainingWindow:i+TrainingWindow+PredictionWindow]
    y_testing = Y[i+TrainingWindow:i+TrainingWindow+PredictionWindow]
    i = i + PredictionWindow
    #training and prediction
    y_rbf = svr_rbf.fit(X_training, Y_training).predict(X_testing)
    for j in y_rbf:
        Y_predicted.append(j)
    for k in X_testing:
        X_predicted.append(k)

    #calculating error (y_true - y_pred)/y_true
    TempError = []
    for m in range(len(y_testing)):
        a = abs((y_testing[m] - y_rbf[m])/y_testing[m])
        if a < 1:
            c = a
        b= ((y_testing[m] - y_rbf[m])/y_testing[m])
        if abs(b) < 1:
            d = b
        TempError.append(c)
        TrueError.append(d)
#converting in an array
    TempError1 = np.asarray(TempError)
#taking mean
    TempMAPE = (TempError1.mean())*100
    MAPE.append((TempMAPE))
    #it is predicting every value in a loop. all predicted values are appended in
    #an array for overall error and graph
    #if error is greater than 10%, decrease the prediction window size
    if TempMAPE > 10:
        PredictionWindow = PredictionWindow -1
        if PredictionWindow == 0:
            PredictionWindow = 1
        PredictionWindowArray.append(PredictionWindow)
    #if error is less then 5%,increase the prediction window
    if TempMAPE < 5:
        PredictionWindow = PredictionWindow +1
```

```
            if PredictionWindow > 3:
                PredictionWindow = PredictionWindow -1
            PredictionWindowArray.append(PredictionWindow)
        else:
            PredictionWindowArray.append(PredictionWindow)

MAPEArray = np.asarray(MAPE)
print ('error is', MAPEArray.mean())
#Extracting data for testing and plotting
Y_testing = Y[TrainingWindow:]
X_testing = X[TrainingWindow:]
model_x, model_y = model()

#plot original/actual data
plt.scatter(X_testing, Y_testing, c='k', label='Actual data')
#plot predicted data
plt.plot(X_predicted, Y_predicted, c='b', label = 'Improved Algorithm')
plt.plot(model_x, model_y, c='#00FFFF', label = 'Conventional SVR')
print ('length of testing and prediction array is', len(Y_testing), len(Y_predicted))
x_step =[]
for i in range(len(PredictionWindowArray)):
    x_step.append(i)
plt.xlim(140,250)
plt.ylim(-20,120)
plt.xlabel('Time (mins)')
plt.ylabel('Average Traffic Speed (km/h)')
plt.legend()

plt.figure(2)
plt.plot(PredictionWindowArray)
plt.xlim(100,200)
#converting in an array
TrueError1 = np.asarray(TrueError)
data = TrueError1
#fitting a normal distribution
#mu, std = norm.fit(data, floc = 0)
df, mu, std = t.fit(data, floc = 0)
mu1, std1 = norm.fit(data, floc=0)
print (' T dist parameteres are', a, mu, std)
print ('Gaussian parameters are', mu1, std1)
```

```
plt.figure(3)
# Plot the histogram.
plt.hist(data, bins=25, normed=True, alpha=0.5)
# Plot the PDF.
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p1 = norm.pdf(x, loc = mu1, scale = std1)
p = t.pdf(x, df, loc = mu, scale = std)
kde_data = data[:, np.newaxis]
X_plot = np.linspace(xmin, xmax, 1000)[:, np.newaxis]
# Gaussian KDE
kde = KernelDensity(kernel='gaussian', bandwidth=0.035).fit(kde_data)
log_dens = kde.score_samples(X_plot)
#plt.plot(X_plot[:, 0], np.exp(log_dens), linewidth = 2)

plt.plot(x, p, 'k', linewidth=2, label = 't-distribution')
plt.plot(x, p1, 'r', linewidth=2, label = 'Gaussian distribution')
title = "Fit results: mu = %.2f,    std = %.2f" % (mu, std)
plt.xlabel('Normalized Error', fontsize = 16)
plt.ylabel('Probability of Error', fontsize = 16)
plt.legend()
plt.show()
```

d. Python Code for Comparison of Different ML Algorithms

```
'''this program compares the performance of proposed model with other ML methods in terms of prediction error'''

import numpy as np
import csv
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.svm import SVR
from scipy.stats import norm, expon, ks_2samp, kstest, t
from pandas import *
from sklearn.neighbors import KernelDensity
#from sklearn.utils import check_arrays
from sklearn import datasets, linear_model
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.preprocessing import StandardScaler

#define window sizes
TrainingWindow = 20
#creating DataFrame object
df = pd.DataFrame()

#reading csv file into DataFrame object
dfa = pd.read_csv('C:/Users/THINKPAD/Desktop/M.Eng Pro/PredictiveAnalytics/03-2018.csv',
parse_dates=['fecha'], index_col='fecha', sep = ';')
dfa = dfa[dfa['identif'] == '90EL69PM01']
#feature. 'intensidad' or 'vmed'
df_intensidad_1 = dfa[['vmed']]

#code for extracting data for particular time
hour = df_intensidad_1.index.hour
selector_new = (( 1 <= hour) & (hour <= 23))
df_total_1 = df_intensidad_1[selector_new]
#interpolation using cubic method for missing values
df_total_1 = df_total_1.resample('5min')
df_total_1 = df_total_1.interpolate(method ='cubic')
hours = df_total_1.index.hour
mins = df_total_1.index.minute
df_total = df_total_1
Y1= df_total.values

#regression method in the library needs array in specific format.
#Converting input and output array in the required format
X=[]
Y=[]
Y11 = []
for i in range(len(df_total)):
    Y11.append(abs(Y1[i][0]))

#getting rid of 0`s and outliers
for i in range(len(Y11)):
    a=Y11[i]+1
    Y.append(a)
for i in range(len(hours)):
    X.append([hours[i], mins[i]])
print (len(X), len(Y))
```

```
X_predicted = []
Y_predicted = []
X_training = X[:3000]
Y_training = Y[:3000]

#initializing different models
svr_rbf = SVR(kernel='rbf', C=1e3, gamma=0.1)
lin_reg = linear_model.LinearRegression()
dec_reg = DecisionTreeRegressor()
rand_for= RandomForestRegressor()

y_rbf = svr_rbf.fit(X_training, Y_training)
y_lin_reg = lin_reg.fit(X_training, Y_training)
y_dec_reg = dec_reg.fit(X_training, Y_training)
y_rand_for = rand_for.fit(X_training, Y_training)
TotalError = []
TrueError = []
x = 3000+(i*200)
X_testing = X[3000:4000]
Y_testing = Y[3000:4000]
a = y_rbf.predict(X_testing)
b = y_lin_reg.predict(X_testing)
c = y_dec_reg.predict(X_testing)
d = y_rand_for.predict(X_testing)
models = [a, b, c, d]

for i in models:
    counter = 0
    TempError = []
    for m in range(len(Y_testing)):
        a = abs((Y_testing[m] - i[m])/Y_testing[m])
        if a < 1:
            counter = counter + 1
            c = a
        else:
            c = 0
        if c > 0:
            TempError.append(c)
    #converting in an array
    TempError1 = np.asarray(TempError)
    print (counter)
```

```
        #taking mean
        TempMAPE = (TempError1.mean())*100
        #print TempMAPE
        TrueError.append(TempMAPE)
        #print TrueError
TotalError.append(TrueError)
print (TotalError)

num_list = []
num_list = TotalError[0]
plt.bar(0, num_list[0], color = 'r')
plt.bar(1, num_list[1], color = 'y')
plt.bar(2, num_list[2], color = 'g')
plt.bar(3, num_list[3], color = 'b')
plt.legend(labels = ['SVM','Linear Reg','Decision Tree','Random Forest'],loc = 'best')
plt.show()
```