Concept Neurons – Handling Drift Issues for Real-Time Industrial Data Mining

Luis Moreira-Matias[1], João Gama[2,4], and
João Mendes-Moreira[2,3]

[1] NEC Laboratories Europe, Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
[2] Faculdade de Economia, U. Porto 4200-465 Porto, Portugal
[3] LIAAD-INESC TEC, 4200-465 Porto, Portugal
[4] DEI-FEUP, U. Porto, 4200-465 Porto, Portugal
luis.moreira.matias[at]gmail.com,
{jgama, joao.mendes.moreira}[at]inescporto.pt

Abstract. Learning from data streams is a challenge faced by data science professionals from multiple industries. Most of them struggle hardly on applying traditional Machine Learning algorithms to solve these problems. It happens so due to their high availability on ready-to-use software libraries on big data technologies (e.g. SparkML). Nevertheless, most of them cannot cope with the key characteristics of this type of data such as high arrival rate and/or non-stationary distributions. In this paper, we introduce a generic and yet simplistic framework to fill this gap denominated Concept Neurons. It leverages on a combination of continuous inspection schemas and residual-based updates over the model parameters and/or the model output. Such framework can empower the resistance of most of induction learning algorithms to concept drifts. Two distinct and hence closely related flavours are presented to handle different drift types. Experimental results on successful distinct applications on different domains along transportation industry are presented to uncover the hidden potential of this methodology.

Keywords: supervised learning, online learning, concept drift, perceptron, stochastic gradient descent, regression, residuals, transportation.

1 Introduction

Today's hype around big data technologies floods the market of professionals with distinct backgrounds and yet a common job role: data scientist. Typically, they are actually very experienced on one of data science related fields (e.g. software engineering). However, they also commonly lack on the theoretical background required to adequately use more than off the shelf Machine Learning techniques and/or methodologies on their daily tasks.

The requirements for a more advanced framework varies naturally from task to task. Hitherto, this issue is more evident when a data mining (DM) task requires **real-time learning.** There are two key issues that empower such fact on these problems: (i) the high sample arrival rate and the constant and/or (ii) bursty drifts on the underlying probability distributions. These characteristics typically disallow the application of most of the traditional Machine Learning techniques (which assume finite training sets and/or stationary distributions) [1].

Recently, some simple approaches to handle this phenomenon have been scatterly introduced on different industries. Two of the most common ones are (i) windowing [2] and (ii) weight-based model selection [3]. The first approach consists into updating our model constantly based on every single arrived sample (i.e. incremental learning) or bunch of the most recent ones. The second one consists on combining multiple models through an weighted average of their outputs based on their recent performance. Although there is a growing interest for this type of methods – followed by successful examples of their inclusion on modern large-scale Machine Learning libraries such as *Mahout* and *SparkML* (e.g. alternating least squares using stochastic gradient descent) - this movement is not certainly keeping up with the explosively increasing speed of industries' needs to answer this particular problem. The most well-known exception is from the recommender systems area and, namely, the winning approach of NetFlix competition: Koren [4] pointed the temporal dynamics and concept drift as one key core ideas of their solution.

This paper intends to fill this gap by promoting a simplistic and yet effective framework that can handle drift on regression problems. Hereby, it is named Concept Neurons. The intuition behind its name comes from the need of a learning schema that can resist to concept drift and/or, *in extremis*, to a total absence of concept (i.e. bursty changes). In the context of predictive modeling in data streams, we have a two-stage (i.e. *predict and correct*) context-aware model [5]: firstly, a predition is made using a given offline/online learner. Secondly, the residuals distribution is monitored with a continuous inspection schema of interest. If a drift alarm is triggered, the prediction's residual is used to update the model whenever possible. Alternatively, we can also update directly the model output for a more bursty reaction to drift.

This schema can cope with most of traditional Machine Learning and/or time series forecasting methods. It was purposely designed on a simple fashion, targeting professionals who have not a strong background on fundamental statistical learning and/or optimization theory. By doing so, we aim to enlarge the pool of practitioners, increase the level of the results of their work as well as the quality of industrial DM pratices in general. Although not bringing a fundamental theoretical contribution, this paper proposes a fully functional idea, simple to understand, to use and with a tremendous applicational potential across industries. Besides the formal description of the present framework, this paper includes two concrete successful examples of their application on the transportation industry – a field where the drift issues are classical problems - including operational control of taxis [6, 7] and of highway networks [8]. Consequently, our contributions are two-fold: (1) to uncover applications of Supervised Learning with drift-handling mechanisms with real-world impact while (2) generalizing a framework that can be adopted by any practitioner on similar problems (from a fundamental point of view), regardless of his/her level of expertise or applicational domain.

The remaining of the manuscript is structured as follows: Section 2 depicts a problem illustration, as well as a brief overview on the related work. The third section formally describes our approach while Section 4 describe the two approach real-world case studies with distinct concept drift natures. Fifth Section describes our experimental test-bed and the obtained results on the abovementioned problems. Finally, conclusions are drawn.

2 Issues on Learning From Non-Stationary Distributions

Real-time DM involves to learn with one (or more) data sources providing samples in a sequential fashion. Typically, this type of data possess unique and complex characteristics to deal with on carrying out Supervised Learning tasks. Some classical examples are high arrival rate, high labeling cost (e.g. [9]) and particularly, **non-stationary distributions**.

The non-stationarity phenomena can be translated in multiple ways. A common scenario is on dealing with datasets containing samples generated from multiple single/joint distributions. Although it is an issue for a vast majority of real-world DM problems (and datasets), it can also be neglected on most of the times to simplify potential paths to their solutions. On the top of the traditional stationarity assumption, many learning algorithms go one step further by assuming a functional form of the dependences and/or a particular residual's distribution (e.g. Gaussian Mixture Models with Expectation-Maximization for clustering; Ordinary Least Squares for regression). Albeit these facts, industrial practitioners rarely test the validity of these assumptions before applying these off-the-shelf Supervised Learning methods. It happens so because this approximation is fairly good for most of the traditional DM problems. Moreover, the trade-off between the time invested on getting alternative solutions and the performance gains often does not pay the effort back. Consequently, a question arises: why should we care about non-stationarity on real-time DM problems?

The main reason to focus on this issue lies on its **timewise** definition. Gama *et al.* [1] characterizes concept drift into four categories: (i) abrupt, (ii) incremental, (iii) gradual and (iv) recurrent. Fig. 1 illustrates a clear example of the latter one using time series data of integers (i.e. highway flow counts). In this particular example, it is somehow safe to assume that the underlying distribution, i.e. $p(y|x)$ is gaussian but for particular days/timespans (e.g. peak hours). This phenomenon is triggered by some sort of exogenous event (e.g. (iv) excessive demand load, (ii) car breakdown or (i) fast weather change) which is unexpected, absent of our data or somehow difficult to model and/or detect beforehand. In many applications, these time periods are actually the critical ones from a business perspective (e.g. peak-hours on transportation, prime time on media, happy hour/discounts on sales/retail).

Three of the most traditional techniques to deal with drift on DM tasks can be enumerated as follows: (1) dynamic model selection (i.e. meta-learning),

(2) windowing and (3) re-training. In (1) model selection, we basically have a bucket of models which are combined dynamically along the time. Two common approaches of this type are weighting models [3] or categorizing samples using a meta-classifier [5]. The first one is simple to understand and to implement as well, being a good answer to (ii) incremental drifts. However, it can arguably deal with (i) abrupt drifts because, typically, the models in the bucket are only periodically updated. A meta-classifer one can handle either (i) abrupt or (iv) recurrent drifts by modeling samples into categories (which have associated labels). Nevertheless, an high level of expertise is required to put such learner in place. On the other hand, (2) windowing can help on dealing either with (ii) incremental and/or (iii) gradual drifts. It consists on considering one or just a bunch of the most recent samples to learn the models [2]. Although being quite simple, this approach is pointed by Gama *et al.* [1] to be slow on detecting (i) abrupt drifts. Model re-training (3) is the most simplistic approach to this problem and one of the most used among industry (e.g. wind power forecasting [10]). Often, it is combined with windowing for engineering-related purposes (e.g., see [11]). Although being pratical and require almost no tuning effort besides the window size, its *blind* reaction to drift - as the model update occurs independently on the samples content - represents a major drawback, thus resulting in a considerable probability of under/overfitting issues.

Our learning schema aims to combine the best of the abovementioned pratices on a simple fashion. The intuition behind it is to provide a very practical mechanism that can be build upon existing and somehow reliable Knowledge Discovery pipelines with proven results to improve their performance even further. The first big advantage on doing so is to re-use the existing DM pipelines (proprietary or not), avoiding costly re-engineering tasks. By leveraging on the existing infrastructure (both physical and intellectual), this framework is easily adoptable by any industrial practitioners facing problems with similar drift-related issues.

3 Concept Neurons

From a high-level perspective, our algorithm operates in two stages: firstly, the residuals distribution produced by a given predictor is monitored by a continuous inspection schema of interest for drift detection purposes. This step aims to assess if the assumptions (here denominated as *Concept*) used to learn it (e.g. stationarity) are being violated. Secondly, a residual-based version of the parameter's inverse gradient is used to update the model whenever possible and/or directly its output. The second stage is only performed whenever an alarm is triggered on the first one, thus activating these updates (here conceptually denoted as *Neuron*).

The present methodology comes in two flavors: (I) asynchronous and (II) synchronous. The first aims on (I-2) (re-)training offline a near-*optimal* explanatory model at regular time intervals and (I-2) keep updating it incrementally in a stochastic fashion using the produced residuals. By extending the offline learning process through an incremental one, we purposely skip the monitoring

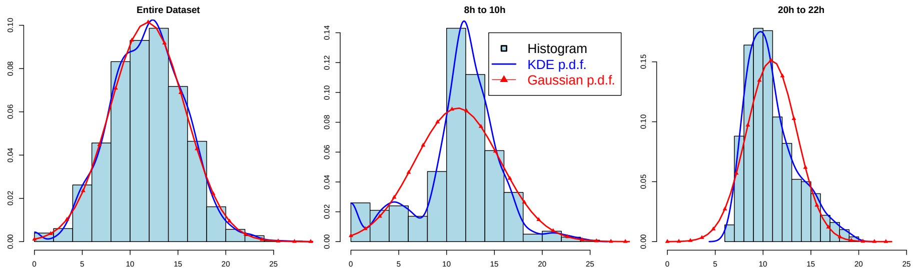

Fig. 1: Timewise drift illustration on a highway flow count data using kernel density estimation (KDE). Globally, the samples approximate a theoretical gaussian density curve. However, this is not true for some day periods due to drifts.

stage by *blindly* assuming that the drift is **constantly happening**. It aims to handle (ii) incremental and (iii) gradual drifts. The second one consists on assuming (II-1) an explanatory learning model (learned either offline or online) to be in place. Then, (II-2) a continuous inspection schema is used to monitor the recent residual's distribution (i.e. windowing) and trigger alarms. Whenever an alarm is triggered, a corrective neuron is activated to start adding up small percentages (i.e. learning rate) of the prediction's **residuals** to our model's output. This rate can be increased as novel alarms are triggered or *deactivated* instead in absence of an alarm for a long period (i.e. here denoted *activation* period). This mechanism aims to handle (iv) recurrent drifts which are limited in time or even bursty ones (when coping with an online learning model). This section describes this methodology fundamentally, departing from its roots in optimization theory till its practical application to Supervised Learning problems.

3.1 Stochastic Learning from Gradients

Let $y_1, ..., y_t : y_i \in \mathbb{R}, \forall i \in \{1..t\}$ denote the values of target variable of interest Y observed till current time t, e.g. train passenger load, and $x_1, ..., x_t : x_i \in \mathbb{R}^n, \forall i \in \{1..t\}$ be the values of an n-dimensional feature matrix $X \in \mathbb{R}^{n \times t}$. Regression problems aim to infer the following function:

$$\hat{f} : x_i, \theta \rightarrow \mathbb{R} \text{ such that } \hat{f}(x, \theta) = f(x_i) = y_i \qquad (1)$$

where $f(x_i)$ denotes the true unknown function which is generating the samples' target variable and $\hat{f}(x_i, \theta) = \hat{y}_i$ be an approximation dependent on the feature vector x_i and an unknown parameter vector $\theta \in \mathbb{R}^n$ (given by some induction model M). Typically, M determines the functional form of $\hat{f}(x_i, \theta)$ as well as the values of θ by formulating a data-driven optimization problem as

$$\hat{f}(x_i, \theta) = \underset{\hat{f}, \theta}{\arg\min} \sum_{i=1}^{t} J(\theta, \hat{f}, x_i, y_i) \qquad (2)$$

where J denotes a cost function of interest and t the number of samples in the dataset. Standard gradient descent is a classical solver. Lets assume that we depart from a given (e.g. random) initialization of our parameter set, i.e. θ_0. The method updates θ iteratively until a certain stopping convergence criteria is met (e.g. ϵ where $\bigtriangledown_\theta > \epsilon$) as follows

$$\theta = \theta - \eta \bigtriangledown_\theta E[J(\theta, \hat{f}, X, Y)] \tag{3}$$

where the above expectation is computed with respect to the abovementioned cost and η denotes our constant learning rate (i.e. an user-defined parameter).

By doing so, we expect to converge to a local minima close enough to our global optimum. Obviously, this does not cope well with an infinite stream of data as our own (i.e. t is being constantly increased $\rightarrow t = +\infty$). A common way to handle this issue is with a stochastic learning (as known as SGD - Stochastic Gradient Descent) of θ. Instead of computing the expectation iteratively, we compute the inverse gradient, i.e. $\bigtriangledown_\theta$ with respect to the most recent labeled sample (x_{t-1}, y_{t-1}), thus redefining recursively the equation 3 as follows

$$\theta_i = \theta_{i-1} - \eta \bigtriangledown_{\theta_{i-1}} J(\theta_{i-1}, \hat{f}, x_{i-1}, y_{i-1}) \tag{4}$$

The cost function most commonly used for regression problems is the well-known l_2 loss. If it is assumed to be in place and for a linear[1] $\hat{f}$, we obtain:

$$J(\theta_i, \hat{f}, x_i, y_i) = L_2(\theta_i, \hat{f}, x_i, y_i) = L_2(\hat{y}_i, y_i) = \frac{1}{2}(y_i - \hat{y}_i)^2 \tag{5}$$

$$\theta_i = \theta_{i-1} - \eta(y_{i-1} - \hat{y}_{i-1})x^T\theta_{i-1} = \theta_{i-1} \cdot \left(1 - \eta(r_{i-1}) \cdot x^T\right) \tag{6}$$

where r_i denotes the prediction's **residual** for sample (x_i, y_i) at time i.

3.2 Asynchronous Concept Neurons

In a real-time context, the simple computation of the $\bigtriangledown_{\theta_i}$ can be problematic (e.g.: missing feature values, noise, $n >> 0$). Therefore, we propose a more naive approach by putting in place the following assumption:

Assumption 1 *Convergence is still possible at a smaller rate when done independently of X for a sufficiently small value of η and an adequate M.*

By doing so, we assume that most of the error is somehow proportional to the values of the parameter set. Formally, we transform equation 6 as follows:

$$\theta_i = \theta_{i-1}(1 - \eta(r_{i-1})) \tag{7}$$

One of the assumptions of SGD is that samples are drawn independently and are identically distributed (i.i.d.). From a theoretical point of view, drift is a violation

[1] Despite the linear assumption (introduced for demonstrative purposes), SGD can also work on non-linear problems departing from a convex loss.

of it. One way of circunventing this issue is to not keep a static learning rate but rather a time-variant one (i.e., $\eta(t)$;, e.g. [12]). The main intuition behind this idea is that the distribution is stationary through a *limited* period of time. Therefore, we can speed up/slow down convergence momentum according to our present learning context.

Departing from this intuition, we introduce a very simple idea in the algorithm 1 based on three simple stages: (1) firstly, learn offline (using M) a model $\hat{f}(\theta, X)$ based on the samples obtained on recent window of time T; (2) Update θ incrementally using the model residuals; (3) re-compute $\hat{f}(\theta, X)$ after T_u periods. T, T_u and η are user-defined parameters and must be tuned for each particular application. Naturally, this approach is expected to handle poorly recurrent and/or bursty drifts as there is no drift detection mechanism embedded.

3.3 Synchronous Concept Neurons

To handle recurrent and/or abrupt drifts, we propose a slight change of the presented learning schema. Intuitively, the idea is that if the concept is dramatically *different*, we do not have time to learn it yet (and consequently, our current model approximation to the target function is quite *poor*). Let $A(R, \phi, \vec{\delta}, t) \in \{0, 1\}$ be a drift detection algorithm of interest where $R = r_1, ..., r_t$ denotes the set of residuals, ϕ denotes a sliding window size, $\vec{\delta}$ stands for generic user parameter set of interest specific for each possible type of A and t the current timestamp. Whenever $A = 1$, the model's output is *corrected* by re-engineering eq. 7 as

$$\hat{y}_i = \hat{f}(x_i, \theta_i) - \eta_i(r_{i-1}) \quad (8)$$

where η is time-dependent from now on, i.e. η_i. If $a_i = 1$, then the learning rate is initialized as $\eta_i = \eta_0$ where η_0 is an initial learning rate set by the user. At this point, we are not fully *trusting* on what $\hat{f}$ is producing as outputs. For most

```
Input: $M$ - offline induction method, $T$ - training window size, $T_u$ - statonarity
       cyclic period; $\eta$ - learning rate, $X, Y$ - dataset;
Output: $\hat{f}$ - approximation function, $\theta_i$ - parameter vector
$W \leftarrow \emptyset$; //Initialization
foreach $i \leftarrow 1..t$ do
    $W \leftarrow W \cup (x_i, y_i)$; // builds offline training set
    if ($T_u \mod i == 0$) then
        $\hat{f}, \theta_i \leftarrow M(W)$; // learns $\hat{f}$ and the parameter set $\theta_i$ from data
    end
    if ($T_u >= t \wedge T_u \mod i > 0$) then
        $\theta_i = \theta_{i-1} - \eta(r_{i-1})\theta_{i-1}$; //update parameter set
        drop an element from the tail of $W$; //forgets outdated samples
    end
end
```

Algorithm 1: Pseudocode for Asynchronous Concept Neurons (ACN).

of applications, it is recommended a conservative approach on the definition of η_i, i.e. $\eta_i << 0$.

Whenever a novel drift occurs, η_i is updated exponentially as $\eta_i = \eta_{i-1}(1+\gamma)$ where $\gamma \in [0, 1]$ denotes a reactivability rate defined by the user. This methodology is not designed to update our models with respect to the observed drift - but rather to *handle* it instead. Intuitevely, in real-time DM problems, if we are facing a recurrent drift, M will be likely to still be useful in the future (as the validity of our current underlying distribution is limited in time). If facing an abrupt drift, this schema will slow down the performance deterioration of the model produced by M but it will not avoid standalone a further (re-)training stage. Consequently, we assume these drifts as time-limited phenomenons. Therefore, the decrease of η_i is operated abruptly as:

$$\sum\nolimits_{i=t-\beta}^{t} A(R, \phi, \vec{\delta}, i) = 0 \qquad (9)$$

In the present context, M can either be an offline or an online induction model. Algorithm (2) depicts the entire schema.

4 Case Studies

Hereby, we approach two different case studies in transportation industry: A) demand prediction for taxi networks and B) road traffic congestion prediction in highway networks. The target clients of the A) are taxi dispatcher's and/or self-organized operators in the sector while B) targets transit authorities and their road traffic management centers.

Case Study A is focused on predicting taxi-passenger demand for short-term horizons of $P-$minutes in a real-time setting [6, 7]. The key idea is to improve the taxi driver's mobility intelligence through a live decision support system advising on best passenger-finding strategy to adopt in each moment (e.g. which is the stand/street/city area that he/she should head to in order to pick up the next passenger).

Case Study B is focused on predicting road Traffic congestion (i.e. incidents). It is possible to divide congestion in two types [8]: (i) *recurrent*, which happens on a regular basis within a given periodicity, e.g. peak hours on every Friday's evening, and a (ii) *stochastic* one, which is provoked by an external event, e.g.: car accidents. The problem is to predict the flow count (number of vehicles that traversed a sensor per unit of time) and occupancy rate (percentage of the time period that a car is over a sensor) on a short-term horizon of $P-$minutes. Then, a scenario-based threshold is considered to transform those discrete signals into binary ones (i.e. congestion/no congestion).

Brief summaries of the datasets are provided below. Additional details about preprocessing tasks conducted over these datasets can be found in Sections 3.2 and 4 of [6] and [8] for case studies A and B, respectively.

```
Input: $\hat{f}$ - approximation function, $\theta$ - parameter set, $\phi$ - monitoring window
       size, $\eta_0$ - initial learning rate, $\gamma$ - constant reactivability rate, $\beta$ -
       activation period, $A$ - drift detection algorithm; $X, Y$-dataset
Output: $\hat{y}$ - corrected predicted outputs
$W \leftarrow \emptyset$ and $\eta_1 \leftarrow 0$;
foreach $i \leftarrow 1..t$ do
    $\eta_i \leftarrow \eta_{i-1}$;
    if ($A(R, T, \vec{\delta}, i) == 1$) then
        if ($\eta_i > 0$) then
            $\eta_i \leftarrow \eta_{i-1}(1 + \gamma)$; //increase the learning rate
        else
            $\eta_i \leftarrow \eta_0$; // activate the prediction corrections
        end
    end
    $\hat{y}_i \leftarrow \hat{f}(x_i, \theta_i) - \eta_i(r_{i-1})$; // correct our prediction output
    $W \leftarrow W \cup (r_{i-1})$; // add elements to the head of $W$
    if ($|W| == T$) then
        drop an element from the tail of $W$;
    end
    if ($\sum_{j=i-\beta}^{i} A(R, \phi, \vec{\delta}, j) == 0$) then
        $\eta_i \leftarrow 0$; // deactivate the prediction corrections
    end
end
```

Algorithm 2: Pseudocode for Synchronous Concept Neurons (SCN).

4.1 (A) Taxi-Passenger Demand Prediction

Our data samples are a stream of timespamped location of events (e.g. pick-up, drop-off) obtained from taxi company (which runs 441 vehicles) operating in Porto, Portugal between August 2011 and April 2012. This city is the center of a medium size urban area with 1.3 million habitants (see Fig. 2).

The drivers operate in 8h shifts: midnight to 8am, 8am-4pm and 4pm to midnight. Each sample arrives has six attributes: (1) TYPE – relative to the type of event reported and has four possible values: *busy* - the driver picked-up a passenger; *assign* – the dispatch central assigned a service previously demanded; *free* – the driver dropped-off a passenger and *park* - the driver parked at a taxi stand. The (2) STOP attribute is an integer with the ID of the related taxi stand. The (3) TIMESTAMP attribute is the date/time in seconds of the event and the (4) TAXI attribute is the driver code; attributes (5) and (6) refer to the LATITUDE and LONGITUDE corresponding to the acquired GPS position.

Table 1 details the number of taxi services demanded per daily shift and day type. Additionally, we can state that the central service assignment is 24% of the total service (*versus* the 76% of the one demanded directly in the street), while 77% of the service demanded directly is dispatched in a stand (and 23% is assigned in cruising time). The average driver waiting time in a stand is 42 minutes while the average cruising time for a service is only $\sim$ 12 minutes.

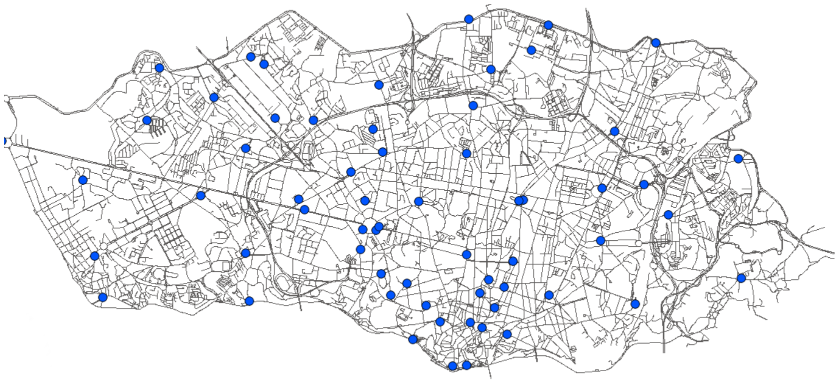

Fig. 2: The spatial distribution of the 63 taxi stands used by this fleet in Porto.

Table 1: Taxi Services Volume (Per Daytype/Daily Shift)

Daytype Group	**Total Services Emerged**	**Averaged Service Demand per Daily Shift**		
		0am to 8am	8am to 4pm	4pm to 0am
Workdays	957265	935	2055	1422
Weekends	226504	947	2411	1909
All Daytypes	1380153	1029	2023	1503

4.2 (B) Highway Congestion Prediction

This dataset was collected through a traffic monitoring system of a major freeway deployed in an Asian country. The studied system broadcasts a stream of traffic-based measurements in real-time with distinct temporal granularities (depending on the type of sensor's installed on each lane). Each sensor measures traffic flow, lane occupancy rate and instantaneous vehicle's speed. The largest time granularity ($p = 5$ minutes) was used to normalize all the collected time series into a standard granularity level.

This network is composed by 106 sensors including both freeway's traffic directions. The covered segment's length is $\sim$ 20km while the sensor's sections are deployed each 500m. Data was collected through 3 non-consecutive weeks.

Fig. 3 depicts an illustration of the dataset. The (B)-figure contains one day of data from a particular section. Conversely, the other chart displays five sample-based p.d.f. obtained using a (gaussian) kernel density estimator over all the flow measurements available - one global and four specific for each of the considered timespans (divided by Periods I-IV, identified by the same display order as Fig. 3 legend). Table 2 details descriptive statistics. The top 10 sensors regarding the number of observed incidents are highlighted. As it is observable, the occupancy rate is higher in these sensors. Not surprisingly. the most critical period is the morning peak (P. II), comprised between 6:40 and 13:20.

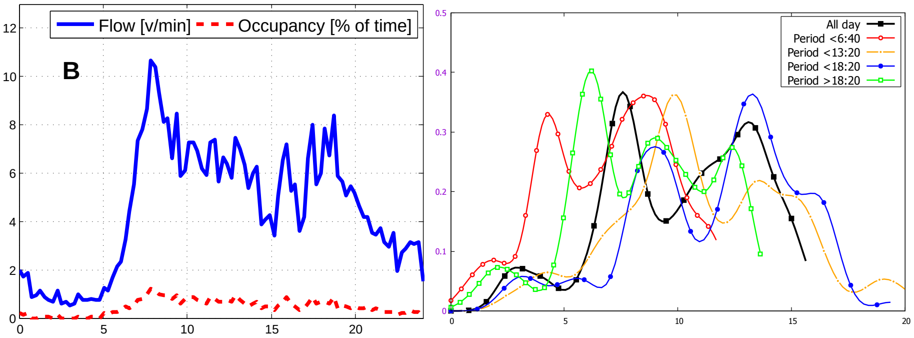

Fig. 3: Illustration of the dataset. The B-figure illustrates data from one-section on one particular day, while the other one depicts a flow-based p.d.f. estimation using all available data.

Table 2: Descriptive Statistics on on all sensors *vs.* top-10 incindent ones.

Quantity	**Flow**				**Occupancy**			
	Mean	**SD**	**Min.**	**Max.**	**Mean**	**SD**	**Min.**	**Max.**
All day	9.9486	3.6514	0.1811	24.4104	2.1409	1.3276	0.0726	13.9967
Top 10, all day	8.2397	3.7001	0.1600	36.7667	3.4303	4.0282	0.0700	23.5567

5 Experiments

On both case studies, we assumed statistical independence among different taxi stands and road sections, respectively. Problem A consists into forecasts one term ahead (i.e. passenger demand count on a specific stand during the next P-minutes). To do it so, we chose a classical method M: Auto Regressive Integrated Moving Averages (ARIMA). For each stand, the ARIMA model was firstly set (and updated each 24 hours) by detecting the underlying model in place from the time series of each stand during the recent $T = 15$ days (i.e. namely, the corresponding $15 \times 2 \times 24 = 1440$ periods). For that purpose, an automatic time series function was employed, i.e. auto-arima [13].

The parameters for each model are generally fit for each period/prediction using a generalized least squares (GLS) solver. Even considering that ARIMA use just a few bunch of recent samples T and low-dimensionality models (i.e. low n), the optimal fitting of its parameters can represent an unnecessary time-consuming process, i.e. $O(N^2)$. In problem A, we can be handling with hundreds of requests on a short amount of time (e.g. 4 different drivers dropping-off a passenger in an interval of two minutes will generate requests to process a total of 252 predictions/GLS - which is equivalent of doing roughly 2,1 model fittings per second on a single CPU) - which raise undesired scalalibility issues. On the top of such computation, as the time series are bounded to the granularity of our forecasting horizon, we have to adapt them in order to obtain the right aggregation level for each on-demand prediction. One way to do it so is to maintain a *newly* calculated discrete time series each τ-minutes where $\tau << P$. By doing so, we can leverage on the additive properties of the series bins (similar to the ones exhibitted by histograms, e.g. [14]) to *roll* our time series into the desired bin positions (e.g. switch from 9:00, 9:30, 10:00,... to 9:10, 9:40, 10:10,...) -, e.g. as proposed by Moreira-Matias *et al.* [6].

To reduce the pratical computational time, we propose to replace GLS by Asynchronous Concept Neurons (ACN). The optimal parameter set θ is fit together with the model estimation stage (i.e. each $T_u = 96$ periods). Then, it is updated as depicted in Algorithm 1. The η value was tuned throughout a grid search procedure in $\big\{0.01k, \forall k \in \{1..20\}\big\}$ using a validation set with data collected on a previous time period.

To approach problem B, we departed from an online learning model which was composed of three main components: (a) an ARIMA-based model, (b) an Exponential Smoothing (ETS) model and (c) an online weighting model to combine both (i.e. ensemble). Similarly to the previous case study, the ARIMA prediction is also performed using a auto-arima+GLS+ACN procedure using $T = 2$ days. However, we are assuming here this schema as a fully incremental method for sake of simplicity. In this case, we decided to test the application of SCN to face the bursty nature of the non-recurrent traffic incidents (e.g. car accidents).

The parameter set θ is composed by both the ARIMA and the ETS model weights, as well as the two weights of each model. The online weighting ensemble were monitoring their performance over a sliding window of H-periods. The drift detection algorithm used was the Page-Hinkley (PH) test, an incremental inspection schema to detect drift [1] (consequently, $\phi = \infty$). The PH test depends on two parameters (i.e. $|\delta| = 2$). In our case, as we are monitoring two series of values (flow and occupancy), we have 4. Their values were set following traffic expert's suggestions. The remaining parameters of this framework η_0, β, γ and also H were tuned using another grid search procedure conducted over six of the 106 sensors of this case study. The full parameter setting employed in our experiments is summarized in Table 3.

5.1 Evaluation

In case study A, we compared traditional ARIMA trained with GLS (ARIGLS) with our ACN using the first as offline baseline (i.e. M). As test set, we considered the last 4 weeks of our data set. Experiments aimed to compare the model's error on it as well the computational time. As evaluation metric, we used an laplacian version of the Symmetric Mean Percentage Error averaged by all the taxi stands. The resulting metric (ASMAPE) is obtained as follows:

$$ASMAPE = \frac{1}{\Upsilon} \sum_{j=1}^{S} \sum_{i=1}^{t} \psi_j \frac{|y_{j,i} - \hat{y}_{j,i}|}{R_{j,i} + X_{j,i} + 1} : \psi_j = \sum_{i=1}^{t} y_{j,i}; \Upsilon = \sum_{j=1}^{S} \psi_j \qquad (10)$$

where S denotes the total number of taxi stands.

In problem B, we compared three distinct online predictive methods: ARIMA (ARI), ETS and the hereby proposed SCN over an online weighted ensemble of both. On the top of the abovementioned sensor selection, we also assumed statistical independence between the data of each one of the three weeks (as they are non-consecutive). Consequently, it resulted on a total of 300 experiments (i.e. 100 sensors x 3 weeks by excluding the 6 sensors used in hyperparameter tuning).

The evaluation of these experiments were performed on two distinct dimensions: (1) numerical prediction and (2) event detection. In (1), we used *Root Mean Squared Error* (RMSE) and *Mean Absolute Error (MAE)* as evaluation metrics. On (2), we picked *Precision* (PRE) and *Recall* (REC). Similarly to A, the results were aggregated using an weighted average of these metrics, where each sensor's weight is given by the total number of incidents occurred.

5.2 Results

The evaluation of two models in case study A are displayed in Table 5. It is possible to observe than, despite their fundamental differences, their performance does not differ significantly. In terms of computational time, ARIGLS took 1.58s to process each individual prediction while ACN took solely 0.60s (in average).

The results for experiments in case B are presented in three distinct folds: Table 4 presents the aggregated results. Left-hand side of Fig. 4 introduces an time-evolving evaluation in terms of RMSE produced by the three flow prediction methods hereby presented. The drift detection(i.e. *neuron activation*) and incident's boolean states are also exhibited on this chart. It is possible to observe that the SCN error is always lower than the one obtained from other methods. On the other hand, we can also conclude than the drift detection is not always necessarily correlated with an incident. The right-hand side of same Fig. 4 llustrates the prediction behavior along sensor with an increasing incident rate (on x-axis). The recall values are averaged using a sliding window considering just the recall values for the latest ten sensors with respect of the current one. By doing so, it is possible to conclude that the our method performance increases along with the number of incidents observed in each sensor.

5.3 Discussion

At a first glance, the high number of hyperparameters may appear a major drawback of our methodology. However, as we could demonstrate, they can be relatively easely tuned with the a validation set. From our experiments, we can sustain that the parameters related with the learning rate (e.g. η in ACN; η_0, β, γ in SCN) are the ones which provoke more variance on the target output. However, it is difficult to assess the framework's sensitivity to the parameter set without a careful evaluation procedure.

In case study A, the results illustrate the computational savings obtained by doing incremental approximations of the optimal model to deal with *soft* drift pheonomenas. In B, the high recall rates are illustrative of the potential of this framework on dealing with either bursty or recurrent drifts.

A work closely related to this one are the Kalman Filters. They are focused on signal processing problems, where our samples are simply a bunch of continuous measurements over time. Conceptually, it also relies on some sort of uncertain estimate/prediction of the series expected value and co-variance to then update it using the residuals co-variance. Formally, we can say that $f(x) = \hat{f}(x, \theta) + v$. Commonly, Kalman Filters assume stationarity on the residuals as $v\ \mathcal{N}(0, \sigma^2)$. Conversely, our approach is fully non-parametric as **it makes any assumption on the residual's distribution**.

In this work, we end up using only linear induction methods as baseline learners for either ACN and SCN. However, **the authors want to highlight that this framework can be built upon non-linear learners as well** - see, for instance, the usage of SCN with decision trees for short-term bus travel time prediction [15]. By being generic and simple to understand as well as to put in pratice, this framework represents a pratical and yet inexpensive alternative to deal with drift on real-world Supervised Learning problems.

6 Final Remarks

Today, experience on Data Science is one of most requested disciplines on job postings across different industries. The lack of qualified professionals on this area with respect to the number of vacancies is biasing companies towards hiring experiencing programmers to then incite them on using off-the-shelf libraries to do *magic* with little developping effort. Hitherto, the availability of drift-aware tools for real-time DM tasks on modern Big Data platforms is scarse. This scenario leads to the misusage of the available tools, poor performance and, ultimately, to reduced business value propositions.

This paper proposes a simple method for handling drift on real-time regression learning problems. It is designed generically, to run on the top of the Supervised Learning schemas popularly employed on modern industrial knowledge discovery pipelines. This two stage framework operates continuously by inspecting the residual's distributions without any predefined assumption on their functional form. Results conducted on real-world trials from the transportation domain demonstrated the potential of this method on reducing computational effort as well as to increase the regressor's generalization error. As future work, we propose to conduct a sensitivity analysis on the parameter setting, as well as to generalize it even more this by introducing an inspection schema able not only to detect drift, but also to categorize its **nature**.

References

1. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Comput. Surv. **46**(4) (2014) 44:1–44:37
2. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. Machine Learning **23**(1) (1996) 69–101

Table 3: Parameter Setting used in the experiments.

		Value	Description
A	P	30	forecasting horizon (in minutes)
	T	1440	training data size (i.e. 15 days)
	T_u	96	size of stationarity cycle period (i.e. 24 hours)
	τ	5	minimum aggregation level (i.e. minutes)
	η	0.01	learning rate
	M	auto-arima + GLS	induction learner
	θ	arima model weights	model's parameter set
	H	4	sliding window size to compute our ensemble
B	A	Page-Hinkley test	drift inspection schema
	P	15	forecasting horizon (in minutes)
	ϕ	∞	drift monitoring window size
	δ_1^f	1.0	max. admissible flow prediction's residual for PH
	δ_2^o	0.1	max. admissible occupancy prediction's residual for PH
	δ_3^f	20	cumulative flow-based threshold to trigger PH alarm
	δ_4^o	4	cumulative occupancy-based threshold to trigger PH alarm
	η_0	0.3	initial learning rate
	β	6	activation period
	γ	0.2	reactivitability rate
	φ_f	10	flow-based min. threshold to trigger an incident
	φ_o	5	occupancy-based max. threshold to trigger an incident

Table 5: Error Comparison on the two Learning Models in A using ASMAPE.

	Periods			
Method	**00h−08h**	**08h−16h**	**16h−00h**	**24h**
ACN	28.47%	24.80%	25.60%	26.21%
ARIGLS	28.23%	24.70%	24.93%	25.80%

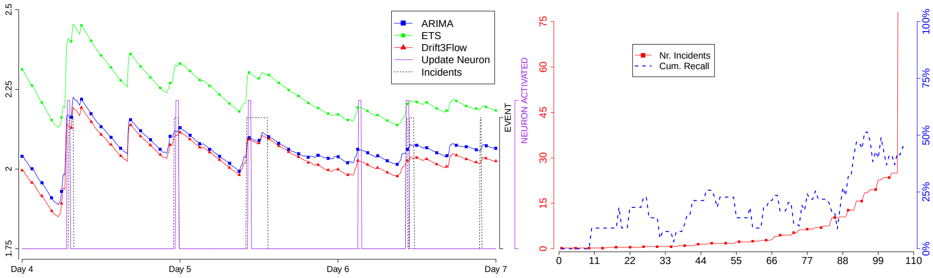

Fig. 4: Illustration of SCN Results: on left-hand side, we have a time-evolving flow-based evaluation on the top-event sensor using RMSE. The right-hand side depicts the average recall for all sensors (on x-axis) ordered by their number of incidents. Note SCN behavior.

Table 4: Results on comparing SCN with ARIMA and ETS in case B.

Method	**Week**	**Flow Prediction**		**Occ. Prediction**		**Event Detection**	
		RMSE	**MAE**	**RMSE**	**MAE**	**PREC**	**REC**
ARI	ALL	1.6875	1.0743	2.1088	1.2939	0.8002	0.2823
ETS	ALL	1.7280	1.0765	2.3111	1.3057	0.8116	0.3000
SCN	ALL	**1.6389**	**1.0379**	**1.8151**	**1.0730**	**0.8199**	**0.3719**

3. Wang, H., Fan, W., Yu, P., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM (2003) 226–235
4. Koren, Y.: Collaborative filtering with temporal dynamics. Commun. ACM **53**(4) (2010) 89–97
5. Žliobaitė, I., Bakker, J., Pechenizkiy, M.: Beating the baseline prediction in food sales: How intelligent an intelligent predictor is? Expert Systems with Applications **39**(1) (2012) 806 – 815
6. Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., Damas, L.: On predicting the taxi-passenger demand: A real-time approach. In: Progress in Artificial Intelligence. Volume 8154 of LNCS. Springer (2013) 54–65
7. Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., Damas, L.: Predicting taxi-passenger demand using streaming data. IEEE Transactions on Intelligent Transportation Systems **14**(3) (2013) 1393–1402
8. Moreira-Matias, L., Alesiani, F.: Drift3flow: Freeway-incident prediction using real-time learning. In: Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on. (Sept 2015) 566–571
9. Žliobaitė, I., Bifet, A., Pfahringer, B., Holmes, G.: Active learning with drifting streaming data. IEEE Transactions on Neural Networks and Learning Systems **25**(1) (Jan 2014) 27–39
10. Monteiro, C., Bessa, R., Miranda, V., Botterud, A., Wang, J., Conzelmann, G., et al.: Wind power forecasting: state-of-the-art 2009. Technical report, Argonne National Laboratory (ANL) (2009)
11. Mendes-Moreira, J., Jorge, A., de Sousa, J., Soares, C.: Comparing state-of-the-art regression methods for long term travel time prediction. Intelligent Data Analysis **16**(3) (2012) 427–449
12. Ikonomovska, E., Gama, J., Džeroski, S.: Learning model trees from evolving data streams. Data mining and knowledge discovery **23**(1) (2011) 128–168
13. Hyndman, R., Koehler, A., Snyder, R., Grose, S.: A state space framework for automatic forecasting using exponential smoothing methods. International Journal of Forecasting **18**(3) (2002) 439–454
14. Gama, J., Pinto, C.: Discretization from data streams: applications to histograms and data mining. In: Proceedings of the 2006 ACM Symposium on Applied Computing, ACM (2006) 662–667
15. Moreira-Matias, L., Gama, J., Mendes-Moreira, J., de Sousa, J.: An incremental probabilistic model to predict bus bunching in real-time. In: Advances in Intelligent Data Analysis XIII. Springer International Publishing (2014) 227–238