Data Mining and Knowledge Discovery manuscript No.
(will be inserted by the editor)

An adaptive algorithm for anomaly and novelty detection in evolving data streams

Mohamed-Rafik Bouguelia · Slawomir Nowaczyk · Amir H. Payberah

Received: date / Accepted: date

Abstract In the era of big data, considerable research focus is being put on designing efficient algorithms capable of learning and extracting high-level knowledge from ubiquitous data streams in an online fashion. While, most existing algorithms assume that data samples are drown from a stationary distribution, several complex environments deal with data streams that are subject to change over time. In this paper, we propose an adaptive method for incremental unsupervised learning from evolving data streams experiencing various types of change. The proposed method maintains a continuously updated network (graph) of neurons by extending the *Growing Neural Gas* algorithm with three complementary mechanisms, allowing it to closely track both gradual and sudden changes in the data distribution. First, an adaptation mechanism handles local changes where the distribution is only non-stationary in some regions of the feature space. Second, an adaptive forgetting mechanism identifies and removes neurons that become irrelevant due to the evolving nature of the stream. Finally, a probabilistic evolution mechanism creates new neurons when there is a need to represent data in new regions of the feature space. The proposed method is demonstrated for anomaly and novelty detection in non-stationary environments. Results show that the method handles different data distributions and efficiently reacts to various types of change.

Keywords Data stream · Growing neural gas · Change detection · Non-stationary environments · Anomaly and novelty detection

1 Introduction

Usual machine learning and data mining methods learn a model by performing several passes over a static dataset. Such methods are not convenient when the data is massive and continuously arriving as a stream. With the big data phenomenon, designing efficient algorithms for incremental learning from data streams is attracting more and more research

Mohamed-Rafik Bouguelia, Slawomir Nowaczyk
Center for Applied Intelligent Systems Research, Halmstad University, Halmstad 30118, Sweden
E-mail: {mohbou, slawomir.nowaczyk}@hh.se

Amir H. Payberah
Swedish Institute of Computer Science, Stockholm, Sweden. E-mail: amir@sics.se

attention. Several domains require an online processing where each data is visited only once and processed as soon as it is available, e.g., due to real-time or limited memory constraints. Applications in dynamic environments experience the so-called *concept drift* [18] where the target concepts or the data characteristics change over time. Such change in streaming data can happen at different speed, being sudden [34, 16] or progressive [8, 22]. Change in streaming data also includes the so called *concept evolution* [32] where new concepts (e.g., classes or clusters) can emerge and disappear at any point in time. Most existing methods, such as those reviewed in [18, 48], address the problem of concept drift and evolution with a focus on supervised learning tasks. In this paper, we focus on online unsupervised learning from an evolving data stream. Specifically, we address the question of how to incrementally adapt to any change in a non-stationary distribution while still perfectly representing the stationary distributions.

The problem is both interesting and important as evolving data streams are present in a large number of dynamic processes [48]. For example, on-board vehicle signals (e.g., air pressure or engine temperature), often used to detect anomalies and deviations [6, 11, 10], are subject to changes due to external factors such as seasons. Other examples include decision support systems in the healthcare domain [33] where advances in medicine lead to gradual changes in diagnoses and treatments, modeling of the human behavior which naturally change over time [45, 37], or the tracking of moving objects on a video [39, 35], to mention but a few.

A naive approach to address the problem of evolving data streams would be to periodically retrain the machine learning model of interest. However, such retraining being triggered without detecting whether it is currently needed or not, often lead to wasted computations. The most widely used approach to deal with changes in data steams consists of training the model based on a *sliding window* [47, 20, 1, 4, 21]. However, choosing a correct window size is not straightforward, since it depends on the speed and type of changes, which are usually unknown. Moreover, existing approaches are specialized for a particular type of change (e.g., sudden, progressive, cyclic). There exist few methods which can handle different types of concept drift, such as [29, 9, 44, 5], however, most of those methods are dedicated for supervised learning problems, where the change is primarily detected by estimating a degradation in the classification performance. Other approaches such as [24, 3] are designed to explicitly detect, in an unsupervised way, when a change happens. Unfortunately, such approaches require hyper-parameters which are hard to set manually when no prior knowledge is available.

Unsupervised neural learning methods such as [14, 38, 41] are good candidates for modeling dynamic environments as they are trained incrementally and take into account relations of neighborhood between neurons (data representatives). Specifically, the *Growing Neural Gas* (GNG) algorithm [14] creates neurons and edges between them during learning by continuously updating a graph of neurons using a *competitive Hebbian learning* strategy [31], allowing it to represent any data topology. This provides an important feature in the context of unsupervised learning from data streams where no prior knowledge about the data is available. However, GNG does not explicitly handle changes in the data distribution.

Some adaptations of GNG such as [13, 41, 30, 15] try to address some of the problems related to either concept evolution [41, 30] or drift [13, 15]. However, these methods require an expert to specify some sensitive parameters that directly affect the evolution or the forgetting rate of the neural network. Setting such global parameters prior to the learning does not address the more general case where the speed of changes can vary over time, or when the distribution becomes non-stationary only in some specific regions of the feature space.

We propose in this paper an extension of the GNG algorithm and we show how it is used for novelty and anomaly detection in evolving data streams. The contributions of this paper are summarized as follows. First, an adaptive learning rate which depends on local characteristics of each neuron is proposed. Such learning rate allows for a better adaptation of the neurons in stationary and non-stationary distributions. Second, a criterion characterizing the relevance of neurons is proposed and used to remove neurons that become irrelevant due to a change in the data distribution. An adaptive threshold for removing irrelevant neurons while ensuring consistency when no change occurs is also proposed. Third, a probabilistic criterion is defined to create new neurons in the network when there is a need to represent new regions of the feature space. The probabilistic criterion depends on the competitiveness between neurons and ensures stabilization of the network's size if the distribution is stationary. The proposed method is adaptive, highly dynamic, and does not depend on critical parameters. It is fully online as it visits each data only once, and can adapt to various types of change in the data distribution.

This paper is organized as follows. In Section 2 we give a background related to the growing neural gas based methods. In Section 3 we propose a mechanism that allows to continuously adapt neurons in order to closely follow any shift in the data distribution. In Section 4 we present an adaptive forgetting mechanism that allows to detect and remove neurons that become irrelevant as a consequence of a change in the data distribution. In Section 5 we present an evolution mechanism that allows to create new neurons when necessary. In Section 6 we summarize the proposed algorithm and we show how it is used for novelty and anomaly detection. In Section 7 we justify the contribution using experimental evaluation. Finally, we conclude and present future work in Section 8.

2 Preliminaries and related work

In this section, we describe the self-organizing unsupervised learning methods that are at the origin of the algorithm proposed in this paper.

The neural gas (NG) [31] is a simple algorithm based on the self-organizing maps [25], which seeks an optimal representation of an input data by a set of representatives called neurons, where each neuron is represented as a feature vector. In this algorithm, the number of neurons is finite and set manually. This constitutes a major drawback, because the number of representatives needed to approximate any given distribution, is usually unknown.

The growing neural gas algorithm (GNG) [14] solves the previous problem by allowing the number of neurons to increase. It maintains a graph G which takes into account the neighborhood relations between neurons (vertices of the graph). As shown in Algorithm 1, a minimum number of neurons is initially created (line 3), then, new neurons and new neighborhood connections (edges) are added between them during learning, according to the input instances. For each new instance x from the stream (line 4), the two nearest neurons n_x^* and n_x^{**} are found (line 6) as follows

$$n_x^* = \underset{n \in G}{\operatorname{argmin}} \|x - n\| ; \qquad n_x^{**} = \underset{n \in G, n \neq n_x^*}{\operatorname{argmin}} \|x - n\| .,$$

where $\|a - b\|$ is the Euclidean distance between vectors a and b. A local representation error $err_{n_x^*}$ is increased for the wining neuron n_x^* (line 7) and the age of the edges connected to this neuron is updated (line 8). The wining neuron (i.e., n_x^*) is adapted to get closer to x, according to a learning rate $\epsilon_1 \in [0, 1]$. The neighboring neurons (linked to n_x^* by an edge) are also adapted according to a learning rate $\epsilon_2 < \epsilon_1$ (line 9). Furthermore, the two neurons

n_x^* and n_x^{**} are linked by a new edge (of age 0). The edges that reached a maximum age a_{max} without being reset, are deleted. If, as a consequence, any neuron becomes isolated, it is also deleted (lines 10-13). The creation of a new neuron is done periodically (i.e., after each λ iterations) between the two neighboring neurons that have accumulated the largest representation error (lines 14-20). Finally, the representation error of all neurons is subject to an exponential decay (line 21) in order to emphasize the importance of the most recently measured errors.

Algorithm 1 Growing Neural Gas (GNG)

1: Input: $\epsilon_1, \epsilon_2, a_{max}, \lambda$
2: $t \leftarrow 0$
3: Initialize graph G with at least 2 neurons
4: **for** each new instance x from the stream **do**
5: $t \leftarrow t + 1$
6: Let n_x^*, n_x^{**} be the two neurons closest to x
7: $err_{n_x^*} \leftarrow err_{n_x^*} + \|x - n_x^*\|^2$
8: Increment the age of n_x^*'s edges
9: Adapt n_x^* and its neighbors (linked to n_x^*)

$$n_x^* \leftarrow n_x^* + \epsilon_1 \times (x - n_x^*)$$

$$\forall n_v \in \text{Neighbours}(n_x^*) : n_v \leftarrow n_v + \epsilon_2 \times (x - n_v)$$

10: **if** n_x^* is linked to n_x^{**}, reset the edge's age to 0
11: **else** Link n_x^* to n_x^{**} with an edge of age 0
12: Remove old edges, i.e., with $age > a_{max}$
13: Remove neurons that become isolated
14: **if** t is multiple of λ **then**
15: Let $n_q = \text{argmax}_{n \in G}\, err_n$
16: Let $n_f = \text{argmax}_{n \in \text{Neighbours}(n_q)}\, err_n$
17: Create a new neuron n_{new} between n_q and n_f
18: $n_{new} = 0.5 \times (n_q + n_f)$
19: $err_{n_{new}} = 0.5 \times err_{n_q}$
20: **end if**
21: Exponentially decrease the representation error of all neurons:

$$\forall n \in G : err_n \leftarrow 0.9 \times err_n$$

22: **end for**

The preservation of neighborhood relations in GNG allows it to represent data typologies of any shape (as shown on Fig. 1), which makes it particularly interesting for a wide range of applications. However, in a non-stationary environment, the GNG algorithm suffers from several drawbacks.

First, it organizes neurons to represent the input distribution by continuously adapting the feature vectors of neurons based on two learning rates ϵ_1, ϵ_2 (see Algorithm 1, line 9), whose values are set manually. If those values are not chosen appropriately, the neural network will not be able to closely follow changes in the data distribution.

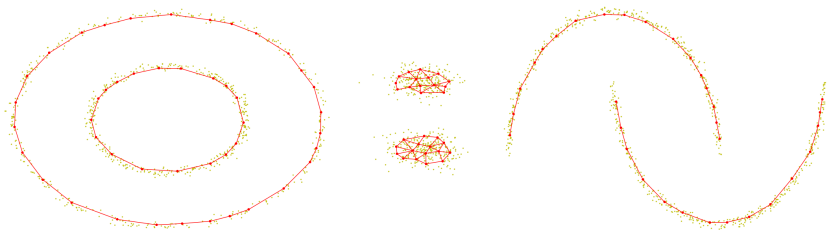

Fig. 1 GNG is able to perfectly learn the topology of data in a stationary environment.

Second, when the distribution changes fast, many neurons will not be updated anymore, and will consequently not be able to follow the change. As such neurons do not become isolated, they will never be removed by GNG. Fig. 2 shows a data distribution which initially forms one cluster and then splits into two clusters. The first cluster, in the bottom left region of Fig. 2 (1), is stationary. The other cluster is moving and getting farther from the first cluster, as shown by the sequence of Fig. 2 (A-D). Neurons that are not able to follow the moving cluster are kept as part of the graph even if they not relevant anymore.

Third, the GNG algorithm suffers from the need to choose the parameter λ (see Algorithm 1, line 14), used to periodically create a new neuron. The periodic evolution of the neural network is clearly not convenient for handling sudden changes where new neurons need to be created immediately. Some adaptations of GNG, like those proposed in [38, 41, 30], try to overcome the problem of periodic evolution by replacing the parameter λ by a distance threshold which can be defined globally or according to local characteristics of each neuron. For each new instance x, if $\|x - n_x^*\|$ is higher than some threshold, then a new neuron is created at the position of x. However, although this overcomes the problem of periodic evolution, setting an appropriate value for the threshold is not straightforward as it highly depends on the unknown distribution of the input data. Moreover, such methods are

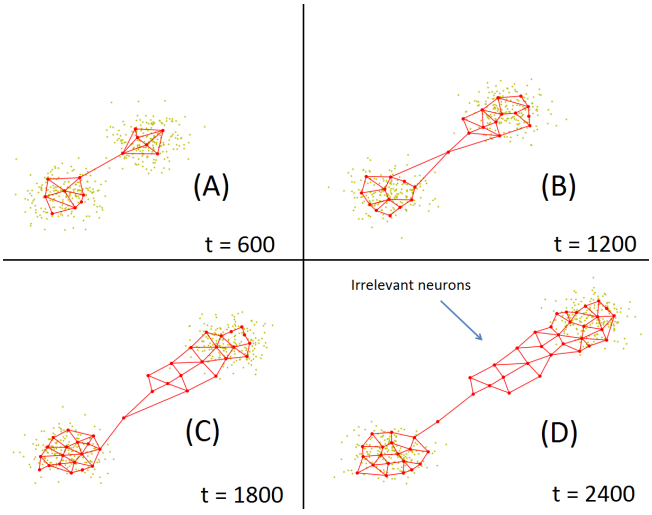

Fig. 2 In GNG, with a non-stationary distribution, some irrelevant neurons are not updated anymore and are never removed.

more prone to representing noise because they create a new neuron directly at the position of x instead of regions where the accumulated representation error is highest (as in the original GNG).

There exist several variants of GNG for non-stationary environments such as [13, 12, 15]. Perhaps the most known variant is GNG-U [15] which is proposed by the original authors of GNG. It defines a utility measure that removes neurons located in low density regions and inserts them in regions of high density. The utility measure is defined as follows. Let n_r be the neuron with the lowest estimated density u_{n_r}. Let n_q be the neuron witch accumulated the highest representation error err_{n_q} (as in line 15 of Algorithm 1). If the ratio $\frac{err_{n_q}}{u_{n_r}}$ is higher than some threshold θ, then the neuron n_r is removed from its current location and inserted close to n_q. However, the threshold θ is yet another user defined parameter which is hard to set because the ratio $\frac{err_{n_q}}{u_{n_r}}$ is unbounded and highly depend on the input data. Moreover, removing n_r and immediately inserting it close to n_q assumes that the distribution is shifting (i.e., the removal and insertion operations are synchronized). Yet, in many cases, we may need to create new neurons without necessarily removing others (e.g., the appearance of a new cluster). Moreover, the evolution of the neural network is still periodic, as in GNG. The only way to limit the network's size is to set a user-specified limit on the number of neurons, which otherwise leads to a permanent increase in the size of the network.

A state of the art method called GNG-T [13], which is an improved version of the method proposed in [12], allows to follow non-stationary distributions by controlling the representation error of the neural network. During an epoch of N successive iterations (i.e., successive inputs to the algorithm), let $\{x_j^i\}_{1 \leq j \leq l_i}$ denote the set of l_i input data for which n_i is the winning neuron. Then the representation error of the neuron n_i is defined as $E_{n_i} = \frac{1}{N} \sum_{j=1}^{l_i} \left\| x_j^i, n_i \right\|$. The method determines the σ-shortest confidence interval [19] (E_{min}, E_{max}) based on the errors of all neurons $\{E_{n_i}\}_{n_i \in G}$. Let T be a target representation error specified by the user. GNG-T seeks to keep the representation error of the neural network close to T by maintaining $T \in [E_{min}, E_{max}]$. More specifically, after each period, if E_{max} becomes less than T, then a neuron is removed. Similarly, if E_{min} becomes higher than T, then a new neuron is inserted. After each epoch, the method simply determines neurons that are no longer relevant as those that have not won, and removes them. GNG-T is the closest work to what we propose in this paper. Unfortunately, it depends on critical parameters (mainly, the epoch N, the target error T and the confidence σ) which directly guide the insertion and removal of neurons. Moreover, splitting the stream into epochs of N successive input data means that GNG-T is only partially online.

In order to relax the constraints related to GNG and its derivatives in non-stationary environments we propose, in the following sections, new mechanisms for: (1) A better adaptation of the neurons in stationary and non-stationary distributions (Section 3); (2) An adaptive removal of irrelevant neurons, while ensuring consistency when no change occurs (Section 4); (3) Creating new neurons when necessary, while ensuring stabilization of the network's size if the distribution is stationary (Section 5).

3 Adaptation of existing neurons

GNG, and the self-organizing neural gas based methods in general, can intrinsically adapt to the slowly changing distributions by continuously updating the feature vector of neurons.

As shown in Algorithm 1 (line 9), this adaptation depends on two constant learning rates ϵ_1 (for adapting the closest neuron n_x^* to the input) and ϵ_2 (for adapting the topological neighbors of n_x^*) such that $0 < \epsilon_2 < \epsilon_1 \ll 1$. If the learning rates ϵ_1 and ϵ_2 are too small, then neurons learn very little from their assigned data. This causes the neural network to adapt very slowly to the data distribution. In contrast, too high learning rates can cause the neural network to oscillate too much.

Many existing methods try to address this problem by decreasing the learning rate over time (also referred to as "annealing" the learning rate), so that the network converges, the same way as it is done for the stochastic gradient descent [46]. However, in a streaming setting, as time goes, this may cause neurons to adapt very slowly to changes in data distribution that happen far in the future (as the learning rate will be very small). Moreover, such a global learning rate is not convenient for handling local changes where the distribution is only stationary in some regions of feature space. Some methods like [41] define the learning rate for the winning neuron n_x^* as being inversely proportional to the number of instances associated with that neuron (i.e., the more it learns, the more it becomes stable). However, such learning rate is constantly decreasing over time, thus still causes neurons to adapt slowly to changes in data distribution as time goes.

In order to closely follow the changing distribution and properly handle local changes, we propose to use an adaptive local learning rate ϵ_n for each neuron n. Intuitively, a local change is likely to increase the local error of nearby neurons without affecting the ones far away. Therefore, we define the learning rate of a neuron n as being proportional to its local error err_n. By doing so, at each point in time, the learning rate of each neuron can increase or decrease, depending on the locally accumulated error.

Let E be the set of local errors for all neurons in the graph G, sorted in the descending order (i.e., from high error to low error). The learning rate for each neuron $n \in G$ is then defined as follows

$$\epsilon_n = \frac{1}{1 + \text{Index}(err_n, E)}, \tag{1}$$

where $\text{Index}(err_n, E)$ is the index (or rank) of err_n in E. Therefore, we define the learning rate used for adapting the winning neuron as $\epsilon_1 = \epsilon_{n_x^*}$, and the learning rate for adapting each neighboring neuron $n_v \in \text{Neighbors}(n_x^*)$ as $\epsilon_2 = \min(\epsilon_1, \epsilon_{n_v})$.

By adapting the wining neurons with a learning rate which is proportional to their local errors, the algorithm manages to better adapt to local changes while still perfectly representing stationary distributions.

4 Forgetting by removing irrelevant neurons

The GNG algorithm can follow a slowly changing distribution by gradually adapting neurons during the learning process. Nonetheless, dealing with concept drifting data implies not only adapting to the new data but also forgetting the information that is no longer relevant. GNG is able to remove neurons that become isolated after removing old edges (lines 12-13 of Algorithm 1). However, as shown previously on Fig. 2, when the data distribution changes sufficiently fast, some neurons will not be adapted anymore and they will still be kept, representing old data points that are no longer relevant. The forgetting mechanism proposed in this section allows us to eliminate such irrelevant neurons in an adaptive way.

4.1 Estimating the relevance of a neuron

In order to estimate the "relevance" of neurons, we introduce a local variable C_n for each neuron n. This local variable allows to ensure that removing neurons will not negatively affect the currently represented data when no change occurs. For this purpose, C_n captures the cost of removing the neuron n. This cost represents how much the total error of the neighboring neurons of n would increase, if n is removed. In order to define C_n, let us consider $X_n = \{x_i \mid n = n^*_{x_i}\}$ as the set of instances (data points) associated with a given neuron n (instances closest to n, i.e., for which n was winner). If n is removed, instances in X_n would be associated to their nearest neurons in the neighborhood of n. Associating an instance $x_i \in X_n$ to its (newly) nearest neuron $n^{**}_{x_i}$ would increase the local error of that neuron by $\|x - n^{**}_{x_i}\|$. Therefore, we define C_n for a neuron n according to the distance from its associated instances to their second nearest neurons, as follows

$$C_n = \sum_{i=0}^{t} \mathbb{1}(n = n^*_{x_i}) \times \left\|x_i - n^{**}_{x_i}\right\|,$$

where t is the current time step (i.e., t'th instance from the stream), and $\mathbb{1}(Cond)$ is the 0-1 indicator function of condition $Cond$, defined as

$$\mathbb{1}(Cond) = \begin{cases} 1 & \text{if } Cond \text{ is true} \\ 0 & \text{otherwise.} \end{cases}$$

In order to compute an approximation of C_n for each neuron n in an online fashion, each time a new instance x is received from the stream, the local variable $C_{n^*_x}$ of its closest neuron n^*_x (i.e., the winning neuron) is increased by $\|x - n^{**}_x\|$ (i.e., by the distance to the second closest neuron)

$$C_{n^*_x} \leftarrow C_{n^*_x} + \left\|x - n^{**}_x\right\|. \tag{2}$$

The local variable C_n is then an estimation for the cost of removing the neuron n. However, C_n as a cost, does not indicate whether the neuron n is irrelevant or not. In order to capture whether a neuron is no longer relevant, the local variable for all the existing neurons is exponentially decreased at each iteration (the same way as it is done for the local representation error in line 21 of Algorithm 1) :

$$\forall n \in G, \quad C_n \leftarrow 0.9 \times C_n. \tag{3}$$

A very small value C_n for a neuron n, may indicate two things. First, when the distribution is stationary, it indicates that n is competing with other neurons in its neighborhood (i.e., the cost of removing n is low), which suggests that the input data associated with n can be safely represented by its neighboring neurons instead. Second, when the distribution is not stationary, it indicates that n is no longer often selected as the closest neuron to the input data, which suggests that it is no longer relevant. In both of those cases, n can be safely removed for a sufficiently small value of C_n.

Let us denote by $\hat{n}$ the neuron which is most likely to be removed (i.e. with the smallest C_n):

$$\hat{n} = \underset{n \in G}{\operatorname{argmin}}\, C_n.$$

Naturally, the forgetting can be triggered by removing $\hat{n}$ if the value of its local variable $C_{\hat{n}}$ falls below a given threshold. However, such value may quickly become small (approaching

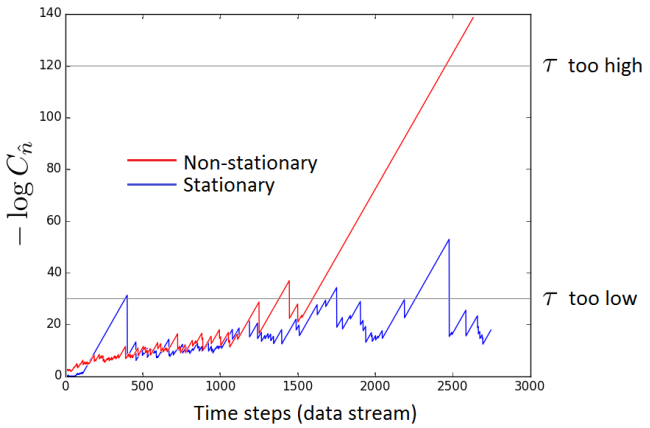

Fig. 3 The value $-\log C_{\hat{n}}$ for the neuron $\hat{n}$ (which is most likely to be removed at each iteration) in two cases: a stationary distribution (blue) and a non-stationary distribution (red).

0) as it is constantly subject to an exponential decay, which makes it hard to directly set a threshold on this value. Instead of removing $\hat{n}$ when $C_{\hat{n}}$ is sufficiently small, a more convenient strategy is to remove it when $-\log C_{\hat{n}}$ is sufficiently high, that is, when

$$-\log C_{\hat{n}} > \tau,$$

where τ is an upper survival threshold. The smaller τ, the faster the forgetting. Larger values of τ would imply a longer term memory (i.e. forgetting less quickly). The exact value of the threshold τ depends, among other factors, on the data distribution and how fast it is changing. This is clearly an unknown information. Therefore, a problem that still remains is: when do we trigger the forgetting mechanism? In other words, how do we decide that C_n is sufficiently small, without requiring the user to directly specify such a sensitive threshold?

In order to have a convenient value for τ, we propose in the following an adaptive threshold.

4.2 Adaptive removal of neurons

In order to motivate the adaptive threshold we propose, let us consider Fig. 3, which shows the value of $-\log C_{\hat{n}}$ for the neuron $\hat{n}$ (i.e., the most likely to be removed at each iteration), for a stationary (blue curve) and a non-stationary (red curve) distributions. Remember that at each time step, the relevance variable is increased for the winning neuron (see Eq. 2), and then exponentially decreased for all neurons in the graph (see Eq. 3). On the one hand, if the neuron $\hat{n}$ has not been selected as winner during some short period of time, then $-\log C_{\hat{n}}$ may temporarily be high, but would decrease again as soon as $\hat{n}$ is updated (see the blue curve on Fig. 3). In this case, if the threshold τ is chosen too low, then it wrongly causes $\hat{n}$ to be immediately removed. On the other hand, if $\hat{n}$ is not anymore selected as winner (in the case of a non-stationary distribution), then $-\log C_{\hat{n}}$ keeps increasing (see the red curve on Fig. 3). In this case, if the threshold τ is chosen too high, then it causes long delay in removing neurons that are not relevant anymore (leading to results similar to

those previously shown on Fig. 2), which is not in favor of a real-time tracking of the non-stationary distribution. In order to automatically adapt the threshold τ, we consider the two following cases:

1. Increasing τ:
   If $-\log C_{\hat{n}} > \tau$ (i.e., $\hat{n}$ should be removed) but $\hat{n}$ would still be winning in the future, then the threshold τ should be increased (to remove less neurons in the future). The reason for increasing τ in this case is that a neuron which should be removed is expected to not be winner anymore (or very rarely) in the future.
2. Decreasing τ:
   If $-\log C_{\hat{n}} \leq \tau$ (i.e., $\hat{n}$ is not removed) but $\hat{n}$ is not winning anymore, then the threshold τ should be decreased (to remove more neurons in the future). The reason for decreasing τ in this case is that a neuron which is not removed is expected to be a winner sufficiently frequently.

To address the first case, when a neuron $\hat{n}$ is removed from G because $-\log C_{\hat{n}} > \tau$, we do not discard it completely; instead, we keep it temporarily, in order to use it for a possible adaptation of the threshold τ. Let R be a buffer (a Queue with FIFO order) where the removed neurons are temporarily kept[1]. Let x be a new instance from the stream, and n^*_x be the nearest neuron to x in G. Let $r^*_x \in R$ be the nearest neuron to x in R. If x is closer to r^*_x than to n^*_x (i.e., $\|x - r^*_x\| < \|x - n^*_x\|$), then r^*_x would have been the winner instead of n^*_x. In this case, we increase τ as follows:

$$\tau \leftarrow \tau + \epsilon \times [(-\log C_{r^*_x}) - \tau], \tag{4}$$

where $\epsilon \in [0, 1]$ is a small learning rate (discussed thereafter in this section).

Finally, we need to design a strategy to maintain the R buffer. Let W_n be the number of times where a neuron n has been winner during the W last time steps (iterations). Let $R' = \{r \in R | W_r = 0\}$ be the subset of neurons from R that has never been a winner during the W last time steps. If $|R'| > k$ (i.e., a sufficient number of neurons are not updated anymore), then we definitively remove the oldest neuron from R.

For the second case, let $|\{n \in G | W_n = 0\}|$ be the number of neurons from G that has never been a winner during the W last time steps. If this number is higher than k and $-\log C_{\hat{n}} \leq \tau$, then we decrease τ as follows:

$$\tau \leftarrow \tau - \epsilon \times [\tau - (-\log C_{\hat{n}})] \tag{5}$$

The learning rate $\epsilon \in [0, 1]$ used in Eq. 4 and 5 for updating τ can be decreased over time, as shown in Eq. 6, so that τ converges more quickly.

$$\epsilon = \frac{1}{1 + N_\tau}, \tag{6}$$

where N_τ is the number of times where τ has been updated (increased or decreased). Alternatively, ϵ can be kept constant if the speed of the changing distribution is expected to change over time (i.e., acceleration is not constant), depending on the application domain.

Besides, in order to give a chance for each neuron in G to be selected as winner at least once, W needs to be at least equal to the number of neurons. Therefore, instead of having a manually fixed value for W, this latter is simply increased if the number of neurons reaches W (i.e. if $|G| \geq W$). Note that in all our experiments W is simply increased by 10 each time $|G| \geq W$.

[1] Note that the local variables of the neurons that we keep in R (except for their feature vectors) are updated at each iteration the same way as the neurons in G.

5 Dynamic creation of new neurons

As explained in Section 2, GNG creates a new neuron periodically. If there is a sudden change in the distribution and data starts to come in new regions of the feature space, the algorithm cannot immediately adapt to represent those regions. This is mainly due to the fact that a new neuron is created only every λ iterations. In many real-time applications, new neurons need to be created immediately without affecting the existing ones (i.e., concept evolution). In order to handle such changes faster, we propose a dynamic strategy that allows creation of new neurons when necessary. The proposed strategy ensures that less neurons are created when the distribution is stationary, while being able to create more neurons if necessary, i.e. when there is a change in the data distribution.

Remember that W_n is the number of times where a neuron n has been winner during the W last time steps (iterations). Let us define the ratio $\frac{W_n}{W} \in [0, 1]$ as the winning frequency of a neuron n. When the number of neurons in G is low, their winning frequency is high. This is essentially due to a low competition between neurons, which gives a higher chance for each neuron to be selected as winner. An extreme case example is when G contains only one neuron which is always winning (i.e. its winning frequency is 1). In contrast, as the number of neurons in G increases, their winning frequency decreases due to a higher competitiveness between neurons. We propose a strategy for creating new neurons in a probabilistic way, based on the current winning frequency of neurons in G.

Let $\mathrm{fq} \in [0, 1]$ be the overall winning frequency in the graph G defined as

$$\mathrm{fq} = \frac{1}{|S|} \times \sum_{n \in S} \frac{W_n}{W}, \tag{7}$$

where S is a set of k neurons from G, with the highest winning frequencies. The higher the overall winning frequency, the higher the probability of creating a new neuron, and vice-versa.

If the data distribution is stationary, then creating new neurons is likely to decrease fq, which implies a smaller probability to create more neurons in the future. However, if there is a change in the data distribution so that new neurons actually need to be created, then fq will automatically increase (which leads to a higher probability of creating more neurons). Indeed, let us assume that data points from a new cluster start to appear. Some existing neurons that are the closest to those points will be selected as winner, making their winning frequencies high, which consequently increases fq. As fq increases, there is more chance for creating new neurons to represent the new cluster. This is illustrated in Fig. 4 which shows fq for a stationary distribution (blue curve) and for a non-stationary distribution (red curve) where a new cluster is suddenly introduced after time step 1000 .

The insertion of a new neuron is done with a probability proportional to fq. However, in order to lower the chance of performing insertions too close in time and give time for the newly inserted neuron to adapt, we introduce a *retarder* term rt defined as follows:

$$\mathrm{rt} = \frac{1}{t - t'},$$

where t is the current time step and $t' < t$ is the previous time when the last insertion of a neuron occurred. Hence, a new neuron is created with a probability $\min(\mathrm{fq} + \mathrm{rt}, 1)$. In other words, a new neuron is created if $\mathrm{rand} > \mathrm{fq} + \mathrm{rt}$, where $\mathrm{rand} \in [0, 1]$ is randomly generated according to a uniform distribution.

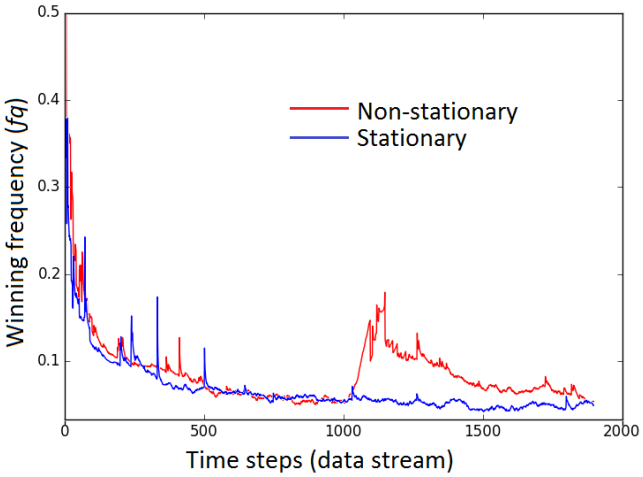

Fig. 4 The value of fq with synthetic data, in two cases: a stationary distribution (blue) and a non-stationary distribution (red).

6 Algorithm

The proposed method is summarized in Algorithm 2, which make a call to Algorithm 3 to check for the removal of neurons (see Section 4) and Algorithm 4 to check for the creation of neurons (See section 5).

First, Algorithm 2 initializes the graph G with two neurons (line 3). For each new data point x, the two nearest neurons n_x^* and n_x^{**} are found (line 7). The local error $err_{n_x^*}$ is updated for the wining neuron n_x^* and the age of the edges emanating from this neuron is incremented (lines 8-9). The local relevance variable $C_{n_x^*}$ is increased in order to keep an information about the cost of removing this neuron (line 10), as described in section 4.1. The neuron n_x^* and its neighbors (linked to n_x^* by an edge) are adapted to get closer to x (lines 11-12), using to local learning rates that are computed according to Eq. 1 as described in Section 3. As in GNG, n_x^* and n_x^{**} are linked by an edge, old edges are deleted, and neurons that becomes isolated are also deleted. Algorithm 3 is called (line 17) to adapt the forgetting threshold τ and to check if there is any irrelevant neuron that needs to be removed, as described in Section 4. Then, Algorithm 4 is called (line 18) in order to insert a new neuron according to a probabilistic criterion described in Section 5. Finally, the local errors and relevance variables of all neurons are subject to an exponential decay (line 19).

Let f be the size of x (i.e. the number of features), g be the number of neurons (i.e. the size of the graph $|G|$), and r be the number of neurons in R, with $r \ll g$. The most time consuming operation in Algorithm 2 is finding the neurons in G that are closest from the input x (line 7 of Algorithm 2). For each input x, this operation takes $\mathcal{O}(f \times g)$. Additionally, the adaptation of the local variables of neurons (e.g. lines 19-22 of Algorithm 2) has $\mathcal{O}(g)$ time complexity. Algorithm 3 has a time complexity of $\mathcal{O}(f \times r)$ as it needs to find r_x^*, and Algorithm 4 has a time complexity of $\mathcal{O}(g)$. Therefore, the overall complexity of the proposed method for learning from each instance x is $\mathcal{O}(f \times (g + r))$, which is very similar to the original GNG algorithm.

Algorithm 2 Proposed method

Input: k (used in sections 4 and 5)
$t \leftarrow 0$ *// current time step*
Initialize graph G with at least 2 neurons
Initialize $\tau > 0$ randomly
for each new instance x from the stream **do**
    $t \leftarrow t + 1$
    Let n_x^*, n_x^{**} be the two neurons closest to x
    $err_{n_x^*} \leftarrow err_{n_x^*} + \|x - n_x^*\|^2$
    Increment the age of n_x^*'s edges
    $C_{n_x^*} \leftarrow C_{n_x^*} + \|x - n_x^{**}\|^2$
    Update the local learning rates according to Eq. 1
    Adapt n_x^* and its neighbors (linked to n_x^*)

$$n_x^* \leftarrow n_x^* + \epsilon_{n_x^*} \times (x - n_x^*)$$

$$\forall n_v \in \text{Neighbours}(n_x^*) : n_v \leftarrow n_v + \epsilon_{n_v} \times (x - n_v)$$

    if n_x^* is linked to n_x^{**}, reset the edge's age to 0
    else Link n_x^* to n_x^{**} with an edge of age 0
    Remove the old edges (as in GNG)
    Remove the neurons that become isolated
    CheckRemoval(k) *// Algorithm 3*
    CheckCreation(k, t) *// Algorithm 4*
    for each $n \in G$ **do**
        $err_n \leftarrow 0.9 \times err_n$
        $C_n \leftarrow 0.9 \times C_n$
    end for
end for

Algorithm 3 CheckRemoval(k)

Let $\hat{n} = \underset{n \in G}{\text{argmin}}\, C_n$; $r_x^* = \underset{r \in R}{\text{argmin}}\, \|x - r\|$; $\epsilon = \frac{1}{1+N_\tau}$ (Eq. 6)

// check if τ need to be increased
if $\|x - r_x^*\| < \|x - n_x^*\|$ and $-\log C_{r_x^*} > \tau$ **then**
    $\tau \leftarrow \tau + \epsilon \times [(-\log C_{r_x^*}) - \tau]$
end if
if $|\{r \in R | W_r = 0\}| > k$ **then**
    Remove (dequeue) the oldest neuron in R
end if

// check if τ need to be decreased
if $|\{n \in G | W_n = 0\}| > k$ and $-\log C_{\hat{n}} \leq \tau$ **then**
    $\tau \leftarrow \tau - \epsilon \times [\tau - (-\log C_{\hat{n}})]$
end if

// check if any neuron need to be removed from G
if $-\log C_{\hat{n}} > \tau$ **then**
    Add (enqueue) $\hat{n}$ to the buffer R
    Remove $\hat{n}$ and its edges from G
    Remove previous neighbors of $\hat{n}$ that become isolated
end if

Algorithm 4 CheckCreation(k, t)

Let S be a set of k neurons in G, with the highest winning frequencies.
$\text{fq} = \frac{1}{|S|} \times \sum_{n \in S} \frac{W_n}{W}$ (see definition of Eq. 7)
if $\underset{uniform}{\text{random}}([0,1]) < \text{fq} + \frac{1}{t-t'}$ **then**
    $t' \leftarrow t$
    Let $n_q = \text{argmax}_{n \in G}\, err_n$
    Let $n_f = \text{argmax}_{n \in \text{Neighbours}(n_q)}\, err_n$
    Create a new neuron n_{new} between n_q et n_f
    $n_{new} = 0.5 \times (n_q + n_f)$
    $err_{n_{new}} = 0.5 \times err_{n_q}$
end if

Anomaly and novelty detection methods [28, 26, 27, 40] learn a model from a reference set of regular (or normal) data, and classify a new test data as irregular (or abnormal) if it deviates from that model. If the reference data comes as a stream and its distribution is subject to change over time, such methods are typically trained over a sliding window as described in [7, 26].

The method we proposed is able to adapt to various types of change without keeping data in a sliding window, and therefore it is straightforward to use it for the task of anomaly and novelty detection where the distribution of the reference data is non-stationary. More specifically, each neuron in G can be considered as the center of a hyper-sphere of a given radius d (a distance threshold). Therefore, at any time t, the graph G (i.e., all hyper-spheres) covers the area of space that represents regular data. It follows that a test data x whose distance to the nearest neuron is less than d, is not part of the area covered by G. More formally, x is considered as abnormal (or novel) if

$$\text{min}_{n \in G}\, \|x, n\| < d$$

Manually choosing a convenient value for d is hard because it not only depends on the dataset but also on the number of neurons in G, which varies over time. Indeed, a higher number of neurons requires a smaller d. However, it is also natural to expect that a higher number of neurons in G would cause the distance between neighboring neurons to be smaller (and vise versa). Therefore, we heuristically set d equal to the expected distance between neighboring neurons in G. In order words, d at any time t is defined as the average length of edges at that time.

7 Experiments

In this section, we evaluate the proposed method. First, we present the datasets used for evaluation in subsection 7.1. Then, we evaluate the general properties of the proposed method in terms of the ability to follow a non-stationary distribution, in subsection 7.2. Finally, we present and discuss the results of the anomaly and novelty detection using the proposed method, in comparison to other benchmarking methods in subsection 7.3.

Table 1 Summary of the datasets characteristics

Dataset	Classes	Features	Size	Change
Keystroke	4	10	1600	200
Sea Concepts	2	3	60000	variable
Usenet	2	658	5931	variable
Optdigits	10	64	3823	NA
1CDT	2	2	16000	400
2CDT	2	2	16000	400
1CHT	2	2	16000	400
2CHT	2	2	16000	400
4CR	4	2	144400	400
4CRE-V1	4	2	125000	1000
4CRE-V2	4	2	183000	1000
5CVT	5	2	40000	1000
1CSurr	2	2	55283	600
4CE1CF	5	2	173250	750
UG-2C-2D	2	2	100000	1000
MG-2C-2D	2	2	200000	2000
FG-2C-2D	2	2	200000	2000
UG-2C-3D	2	3	200000	2000
UG-2C-5D	2	5	200000	2000
GEARS-2C-2D	2	2	200000	2000
2D-1	2	2	5000	NA
2D-2	2	2	5000	NA
2D-3	3	2	5000	NA
2D-4	3	2	5000	NA

7.1 Datasets

We consider in our experimental evaluation several real-world and artificial datasets covering a wide range of non-stationary distributions. Table 1 gives a brief summary of all the considered datasets. The column *Classes* indicates the number of classes or clusters in each dataset. The column *Change* indicates the interval, in number of examples, between consecutive changes (note that NA refers to "no change").

The first group consists in four real world datasets: *Keystroke*, *Sea Concepts* and *Usenet* (which are publicly available for download[2]), and *Opendigits* (which is publicly available at [2]). The *Keystroke* dataset (described in [42]) is from a real-world application related to keystroke dynamics for recognizing users by their typing rhythm, where user profiles evolve incrementally over time. The *Sea concepts* dataset is adapted for novelty detection and originally proposed by [43]. The *Usenet* dataset is a text dataset inspired by [23] and processed for novelty detection. It consists of a simulation of news filtering with concept drift related to the change of interest of a user over time. A user can decide to unsubscribe from news groups that he is not interested in and subscribe for new ones that he becomes

[2] Publicly available for download at `http://www.liaad.up.pt/kdus/products/datasets-for-concept-drift`

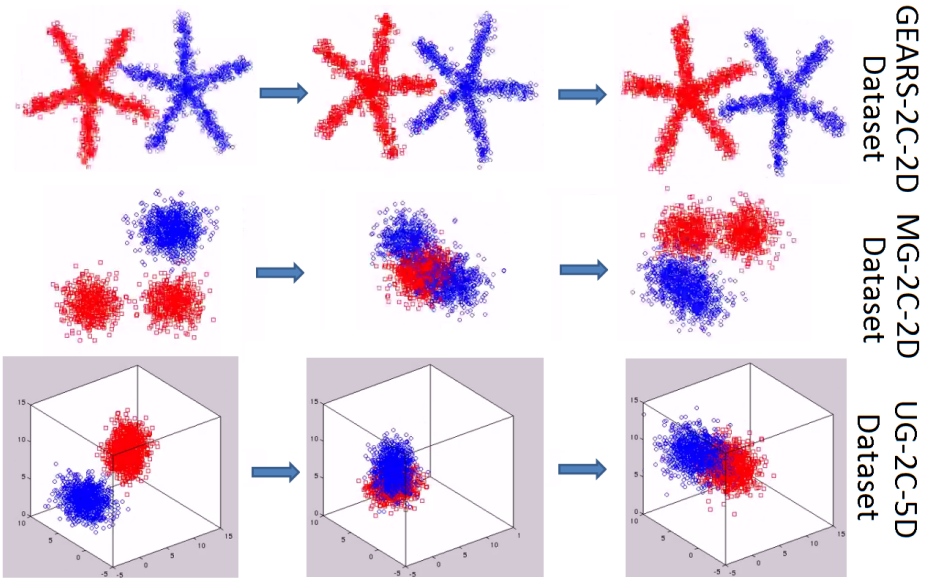

Fig. 5 Some example snapshots from three artificial non-stationary datasets.

interested in, and the previously interesting topics become out of his main interest. The *Optdigits* dataset is a real-word stationary dataset, described in the UCI machine learning repository [2]. It is introduced to test the performance of the proposed method in a stationary environment, where the data is split across multiple clusters.

The other datasets are provided in [42] and publicity available for download[3]. These datasets experience various levels of change over time, thus, are ideal to showcase the performance of algorithms in non-stationary environments. Three examples of these datasets are illustrated in Fig. 5. The last four artificial datasets in Table 1 are stationary datasets with distributions corresponding to various shapes, similar to those previously shown in Fig. 1.

7.2 General properties of the proposed method

The first set of experiments shows the general properties of the proposed method in terms of the adaptability, the ability to represent stationary distributions, and to follow non-stationary distributions.

As an initial step, Fig. 6 illustrates a simple proof of concept of the proposed method for a simulated one-dimensional non-stationary data distribution, which is initially shown by the grey box on the left. The location, over time, of the created neurons is shown with red points. The size of the graph is limited to 10 for better visualization purposes. At time 1000, the distribution is moderately shifted, which makes half of the neurons to be reused, and others to be created. At time 3000 the distribution suddenly changes, which makes only few neurons change their location, and leads to the creation of many new neurons and the

[3] The non-stationary artificial datasets are publicly available for download at `https://sites.google.com/site/nonstationaryarchive/`, where animated visualization of the data over time are also available.

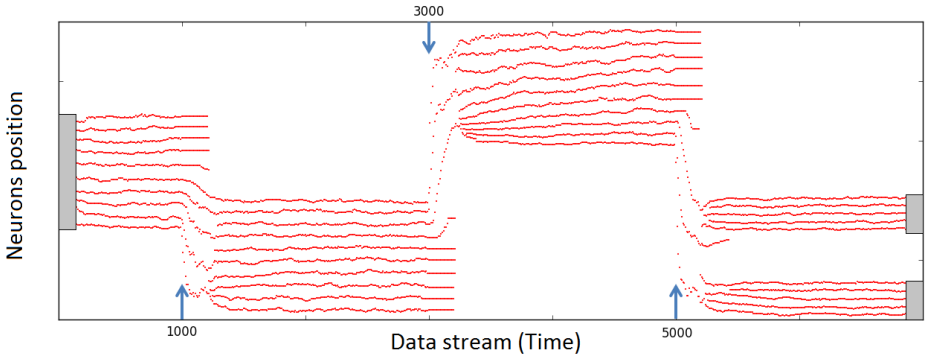

Fig. 6 The location of neurons for a one-dimensional non-stationary distribution over time. $|G| \leq 10$, $k = 10$.

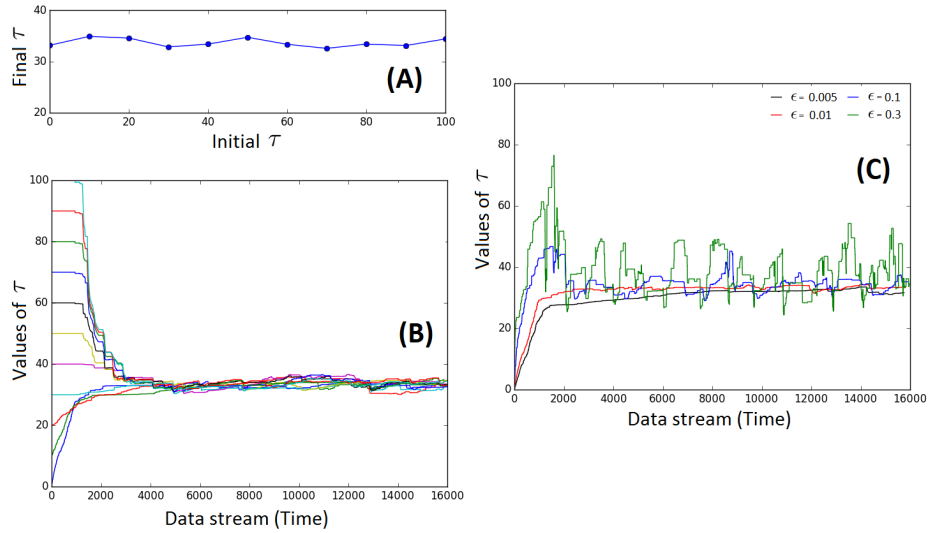

Fig. 7 (A) and (B) show the value of the adaptive threshold τ according to different initial values. (C) shows the effect of ϵ on the adaptation of the threshold τ. Parameter $k = 10$.

removal of existing ones. After time 5000, the distribution splits into two parts (clusters), and the proposed method follows the change.

Fig. 7 shows an experiment performed on the *1CDT* dataset (non-stationary) with the goal to showcase the adaptive removal of neurons described in section 4.2. Remember that, for a given dataset, the forgetting rate depends on the value to which the threshold τ is set. As τ is adaptive, we show in this experiment that it does not depend on the initial value to which it is set. Fig. 7 (A) shows the final values of τ (i.e., after adaptation) according to the initial values of τ. We can see that for any initial value of τ, it always converges to values that are close to each other. Fig. 7 (B) shows the current value of τ over time, based on different initial values. We can see that for any initial value, it converges quickly, i.e., without requiring a significantly large number of adaptation steps. This proves that the method is insensitive to initial values of τ. Moreover, a learning rate ϵ is used during the

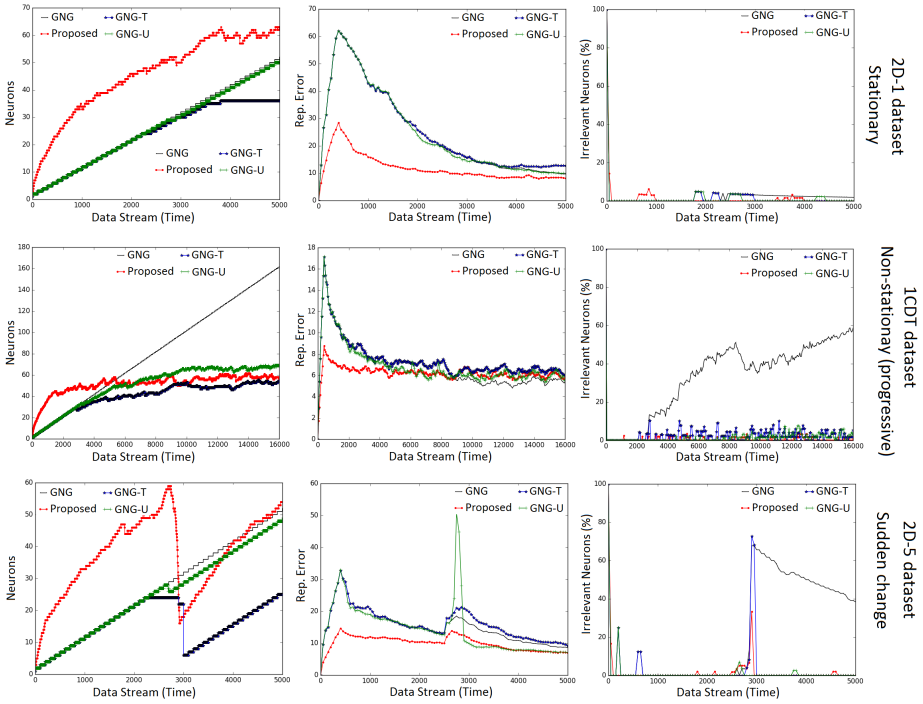

Fig. 8 The period $\lambda = 100$ in GNG, GNG-U and GNG-T. The removal threshold $\theta = 10^9$, 10^5 and 10^7 for the three respective datasets in GNG-U. The epoch $N = 500$, the confidence $\sigma = 0.8$, and target error $T = 0.3$, 0.01 and 0.4, for the three respective datasets in GNG-T. $k = 10$ for the proposed method.

adaptation of τ (see Eq. 5 and 4). In the experiment of Figs. 7 (A) and (B), this learning rate was set according to Eq. 6 as described in section 4.2. In order to illustrate the effect of ϵ on τ, Fig. 7 (C) shows the value of τ over time, based on different values of ϵ. It is natural that when ϵ is low (e.g. 0.005), τ changes slowly at each adaptation step. For τ to eventually stabilize, ϵ needs only to be sufficiently small. Nonetheless, for the reminder of this paper, ϵ is adapted according to Eq. 6.

Fig. 8 illustrates the behavior of proposed method in comparison to GNG [14] and two other variants for non-stationary distributions described in section 2, namely GNG-U [15] and GNG-T [13]. For each method, we show the evolution of the number of neurons over time in first column of figures, the overall representation error (i.e., average over all neurons) in the second column of figures, and the percentage of irrelevant neurons (i.e., that have never been updated during the last 100 steps) in the third column of figures. We show the results in three situations: a stationary distribution using dataset *2D-1* (the three top figures), a non-stationary distribution with a progressive change using dataset *1CDT* (the three middle figures), and a stationary distribution with a sudden change happening after time 2500, suing dataset *2D-4* (the three bottom figures). We can observe from Fig. 8 that the proposed method manages to create more neurons at early stages, which leads to a lower representation error. The number of neurons automatically stabilizes over time for the proposed method (unlike GNG and GNG-U). It also stabilizes for GNG-T depending on the user specified target parameter T. Moreover, all three methods (unlike GNG) efficiently remove irrelevant neurons. Nonetheless, it should be noted that the proposed method is adaptive,

Table 2 Parameters used for Isolation Forest and OCSVM.

Datasets	**Parameters**	
	Isolation Forest	OCSVM
Keystroke	$\text{ø} = 0.2$	$\gamma = 0.2, \nu = 0.3$
Sea Concepts	$\text{ø} = 0.1$	$\gamma = 0.1, \nu = 0.1$
Usenet	$\text{ø} = 0.2$	$\gamma = 0.1, \nu = 0.1$
Optdigits	$\text{ø} = 0.2$	$\gamma = 0.1, \nu = 0.2$
1CDT	$\text{ø} = 0.05$	$\gamma = 0.5, \nu = 0.05$
2CDT	$\text{ø} = 0.1$	$\gamma = 0.5, \nu = 0.2$
1CHT	$\text{ø} = 0.05$	$\gamma = 0.5, \nu = 0.01$
2CHT	$\text{ø} = 0.3$	$\gamma = 0.5, \nu = 0.4$
4CR	$\text{ø} = 0.05$	$\gamma = 0.5, \nu = 0.01$
4CRE-V1	$\text{ø} = 0.05$	$\gamma = 0.5, \nu = 0.01$
4CRE-V2	$\text{ø} = 0.05$	$\gamma = 0.5, \nu = 0.01$
5CVT	$\text{ø} = 0.05$	$\gamma = 0.5, \nu = 0.01$
1CSurr	$\text{ø} = 0.05$	$\gamma = 0.5, \nu = 0.01$
4CE1CF	$\text{ø} = 0.05$	$\gamma = 0.5, \nu = 0.01$
UG-2C-2D	$\text{ø} = 0.05$	$\gamma = 0.5, \nu = 0.01$
MG-2C-2D	$\text{ø} = 0.05$	$\gamma = 0.5, \nu = 0.01$
FG-2C-2D	$\text{ø} = 0.05$	$\gamma = 0.5, \nu = 0.01$
UG-2C-3D	$\text{ø} = 0.05$	$\gamma = 0.3, \nu = 0.01$
UG-2C-5D	$\text{ø} = 0.05$	$\gamma = 0.2, \nu = 0.01$
GEARS-2C-2D	$\text{ø} = 0.1$	$\gamma = 0.5, \nu = 0.2$
2D-1	$\text{ø} = 0.1$	$\gamma = 0.5, \nu = 0.01$
2D-2	$\text{ø} = 0.1$	$\gamma = 0.5, \nu = 0.01$
2D-3	$\text{ø} = 0.05$	$\gamma = 0.4, \nu = 0.01$
2D-4	$\text{ø} = 0.05$	$\gamma = 0.4, \nu = 0.05$

and unlike GNG-U and GNG-T, there is no need to adapt any parameter across the three datasets[4].

7.3 Anomaly and novelty detection

In the following, the proposed method is compared against two anomaly and novelty detection methods, namely One Class SVM (OCSVM) [40] and Isolation Forest [28] which are trained over a sliding window, allowing them to handle non-stationary distributions. The Python implementations available on the scikit-learn machine learning library [36] have been used . A sliding window of 500 instances is chosen for OCSVM and Isolation Forest, as it provides the best overall results across all datasets. For each dataset, instances from a subset of classes (half of the number of classes) are considered as normal (or regular) instances, and the instances from the other half are considered as abnormal (or novel). The considered methods are trained based on the stream of regular instances. As advocated by [17], a prequential accuracy is used for evaluating the performance of the methods in correctly distinguishing the regular vs. novel instances. This measure corresponds to the average accuracy computed online, by predicting for every instance whether it is regular or

[4] The parameter k of the proposed method is always fixed to $k = 10$ for all the experiments and datasets.

Table 3 Average accuracy for distinguishing normal and abnormal (or novel) data

Datasets	**Average accuracy (%)**			**P value (t-test)**
	Proposed	Isolation Forest	OCSVM	
Keystroke	**71.19** $\pm$ 2.75	70.92 $\pm$ 3.53	59.17 $\pm$ 1.55	0.89
Sea Concepts	**78.62** $\pm$ 0.21	75.12 $\pm$ 0.26	75.0 $\pm$ 0.32	9.4e$-$80
Usenet	84.37 $\pm$ 0.44	75.13 $\pm$ 0.36	**84.53** $\pm$ 0.40	0.59
Optdigits	**91.48** $\pm$ 1.44	78.74 $\pm$ 0.80	89.33 $\pm$ 0.31	0.0043
1CDT	**97.47** $\pm$ 0.41	97.24 $\pm$ 0.28	97.10 $\pm$ 0.40	0.368
1CHT	**96.68** $\pm$ 0.66	95.64 $\pm$ 1.12	87.62 $\pm$ 4.07	0.115
1CSurr	92.73 $\pm$ 0.57	92.44 $\pm$ 0.55	**92.79** $\pm$ 0.66	0.899
2CDT	86.98 $\pm$ 1.02	**89.74** $\pm$ 0.54	88.15 $\pm$ 0.33	5.2e$-$6
2CHT	74.90 $\pm$ 1.57	**78.83** $\pm$ 0.49	68.86 $\pm$ 0.53	4.9e$-$6
5CVT	86.60 $\pm$ 0.60	**87.35** $\pm$ 0.51	74.12 $\pm$ 0.53	0.065
4CR	97.08 $\pm$ 0.04	97.28 $\pm$ 0.11	**99.36** $\pm$ 0.01	$\simeq$ 0.0
4CE1CF	**96.99** $\pm$ 0.05	96.48 $\pm$ 0.07	91.46 $\pm$ 0.24	4.4e$-$25
4CRE-V1	**94.01** $\pm$ 0.63	91.99 $\pm$ 0.78	52.15 $\pm$ 0.15	8.5e$-$5
4CRE-V2	**85.48** $\pm$ 0.83	79.16 $\pm$ 0.82	50.61 $\pm$ 0.04	1.5e$-$25
FG-2C-2D	**88.02** $\pm$ 0.67	83.63 $\pm$ 0.90	82.57 $\pm$ 1.17	3.2e$-$14
MG-2C-2D	87.33 $\pm$ 0.53	**87.56** $\pm$ 0.52	79.29 $\pm$ 0.62	0.540
UG-2C-2D	92.26 $\pm$ 0.48	**92.28** $\pm$ 0.50	89.10 $\pm$ 0.71	0.968
UG-2C-3D	**88.28** $\pm$ 0.58	85.70 $\pm$ 0.78	84.51 $\pm$ 0.95	2.3e$-$7
UG-2C-5D	**84.27** $\pm$ 0.39	78.85 $\pm$ 0.65	73.42 $\pm$ 0.71	5.2e$-$42
GEARS-2C-2D	**97.46** $\pm$ 0.04	93.78 $\pm$ 0.09	85.91 $\pm$ 0.11	$\simeq$ 0.0
2D-1	**98.02** $\pm$ 4.08	90.57 $\pm$ 2.44	49.47 $\pm$ 0.08	0.002
2D-2	**99.46** $\pm$ 0.17	94.69 $\pm$ 0.25	86.34 $\pm$ 0.58	4.3e$-$34
2D-3	95.95 $\pm$ 1.13	94.82 $\pm$ 3.61	**99.48** $\pm$ 0.10	2.2e$-$7
2D-4	95.45 $\pm$ 0.82	**96.62** $\pm$ 1.59	90.62 $\pm$ 5.70	0.175

novel, prior to its learning. The average accuracy is estimated using a sliding window of 500 instances.

Table 2 lists, for each dataset, the parameters selected for Isolation Forest and OCSVM, which leaded to the best results. The parameter $\emptyset$ of Isolation Forest refers to the amount of contamination of the data set (i.e., the proportion of outliers in the data set). This parameter is used when training to define the threshold on the decision function. The respective parameters γ and ν of OCSVM refer to the kernel coefficient for the radial basis function (RBF) and to the upper bound on the fraction of training errors.

To present more complete results, the accuracy of the novelty detection over time is detailed in Figs. 9, 10, and 11, for all methods on each datasets. However, for more clarity, the results are summarized in Table 3.

Table 3 show the overall results by presenting the average accuracy over time, as well as the the p-value obtained based on the Student's t-test. This p-value indicates how much significantly the results of the proposed method differ from the results of the best performing method (among OCSVM and Isolation Forest). For each dataset, the result of the best performing method is highlighted with bold text in Table 3. If the result of the best performing

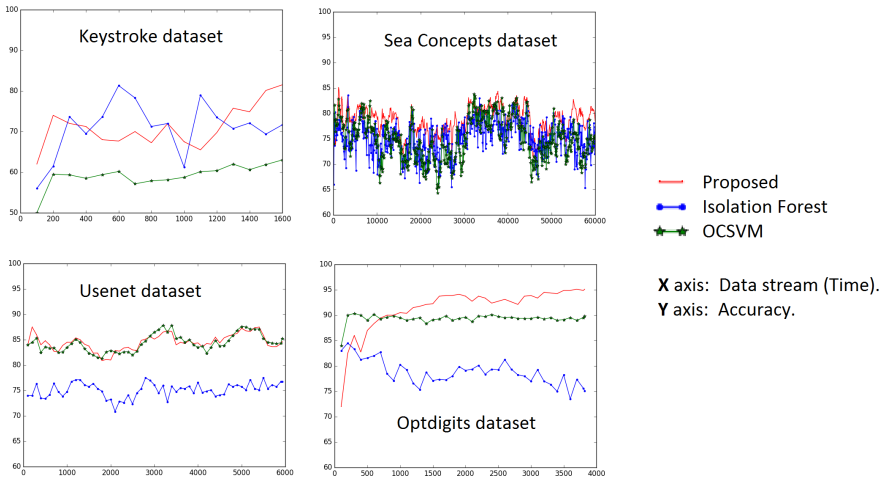

Fig. 9 Results of the novelty detection on real-world datasets

method is not significantly different from the other methods (i.e. p-value > 0.05) then the result those methods is also highlighted with an underlined text.

It can be seen from Table 3 that the proposed method achieves a better overall performance than the other methods. Moreover, as indicated by the p-values, in the vast majority of cases, the results achieved by the proposed method are significantly different from the results achieved by the other methods. These results confirm that the adaptive proposed method is generally more convenient for various types of non-stationary environments, due to its dynamic adaptation over time and its ability to represent data topologies of any shape.

8 Conclusion and future work

In this paper we have introduced a new method for online learning from evolving data streams for the task of anomaly and novelty detection. The method extends GNG for a better adaptation, removal and creation of neurons. The usefulness of the proposed method was demonstrated on various real-world and artificial datasets covering a wide range of non-stationary distributions. The empirical and statistical evaluation that we performed show that the proposed method achieved the best overall performance compared to two state of the art methods, while being much less sensitive to initialization parameters. Results show that the proposed adaptive forgetting and evolution mechanisms allow the method to deal with any change in the non-stationary distribution while still perfectly representing the distribution when it is stationary. Hence, the proposed method is convenient for a wide range of application domains varying from static environments to highly dynamic ones.

For future work, we plan to design a parallel version of the proposed method and take advantage of data intensive computing platforms, such as *Apache Spark*, for the implementation. In order to do this, one way is to parallelize independent parts of the data and process them in parallel while sharing the same graph of neurons. Another alternative is to design algorithms to adequately split and distribute the graph of neurons on multiple machines running in parallel.

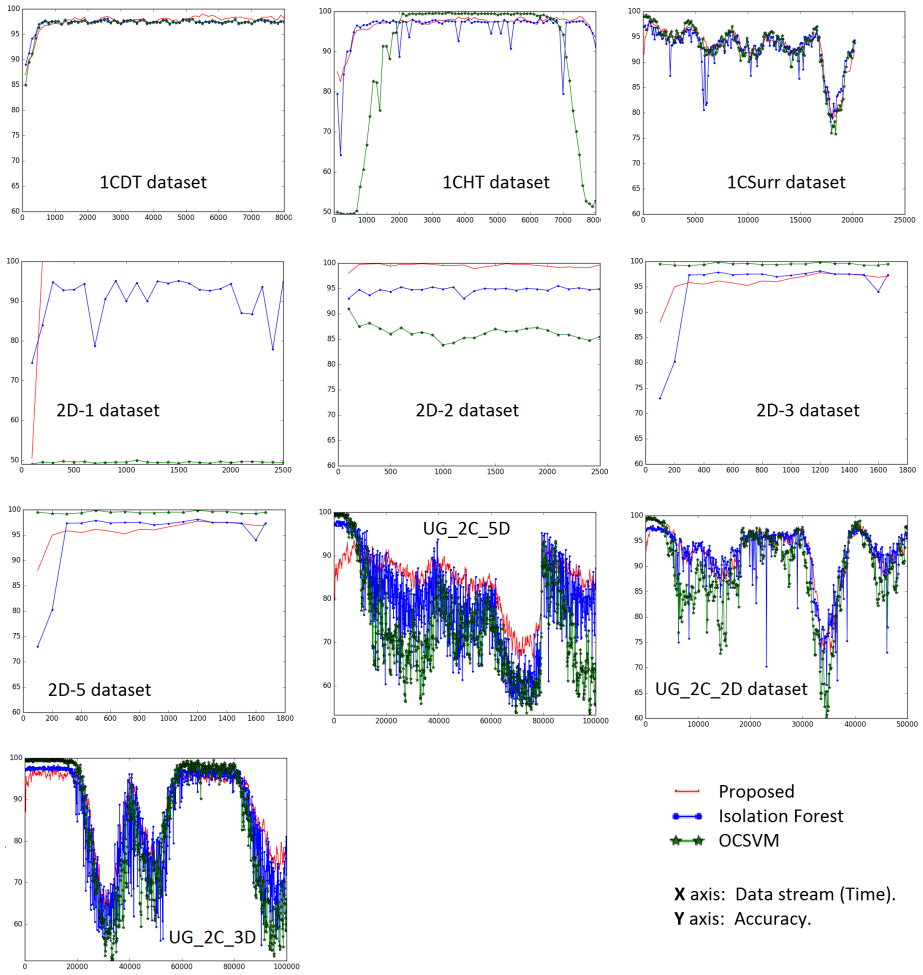

Fig. 10 Results of the novelty detection on artificial datasets

References

1. Ahmed, E., Clark, A., Mohay, G.: A novel sliding window based change detection algorithm for asymmetric traffic. IFIP International Conference on Network and Parallel Computing (2008)
2. Bache, K., Lichman, M.: Uci machine learning repository. http://archive.ics.uci.edu/ml. Irvine, CA : University of California, School of Information and Computer Science (2013)
3. Bifet, A.: Adaptive stream mining: Pattern learning and mining from evolving data streams. Proceedings of the 2010 conference on adaptive stream mining: Pattern learning and mining from evolving data streams pp. 1–212 (2010)
4. Bifet, A., Gavalda, R.: Learning from time-changing data with adaptive windowing. International Conference on Data Mining pp. 443–448 (2007)

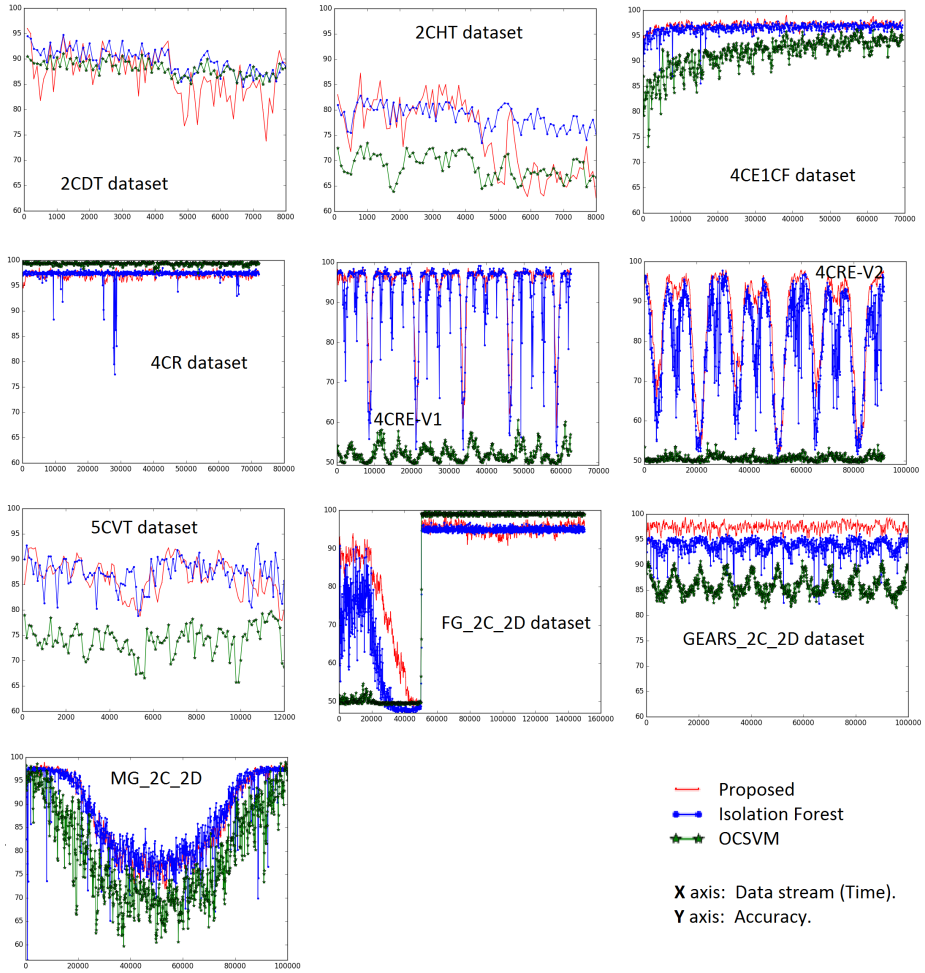

Fig. 11 Results of the novelty detection on artificial datasets

5. Brzezinski, D., Stefanowski, J.: Reacting to different types of concept drift: The accuracy updated ensemble algorithm. IEEE Transactions on Neural Networks and Learning Systems **25(1)**, 81–94 (2014)
6. Byttner, S., Rognvaldsson, T., Svensson, M.: Consensus self-organized models for fault detection (cosmo). Engineering Applications of Artificial Intelligence **24(5)**, 833–839 (2011)
7. Ding, Z., Fei, M.: An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. IFAC Proceedings **46(20)**, 12–17 (2013)
8. Ditzler, G., Polikar, R.: Incremental learning of concept drift from streaming imbalanced data. IEEE Transactions on Knowledge and Data Engineering **25(10)**, 2283–2301 (2013)
9. Dongre, P., Malik, L.: A review on real time data stream classification and adapting to various concept drift scenarios. IEEE International Advance Computing Conference pp. 533–537 (2014)

10. Fan, Y., Nowaczyk, S., Rognvaldsson, T.: Evaluation of self-organized approach for predicting compressor faults in a city bus fleet. Procedia Computer Science **53**, 447–456 (2015)
11. Fan, Y., Nowaczyk, S., Rognvaldsson, T.: Incorporating expert knowledge into a self-organized approach for predicting compressor faults in a city bus fleet. Scandinavian Conference on Artificial Intelligence pp. 58–67 (2015)
12. Frezza-Buet, H.: Following non-stationary distributions by controlling the vector quantization accuracy of a growing neural gas network. Neurocomputing **71(7)**, 1191–1202 (2008)
13. Frezza-Buet, H.: Online computing of non-stationary distributions velocity fields by an accuracy controlled growing neural gas. Neural Networks **60**, 203–221 (2014)
14. Fritzke, B.: A growing neural gas network learns topologies. Advances in neural information processing systems **7**, 625–632 (1995)
15. Fritzke, B.: A self-organizing network that can follow non-stationary distributions. International Conference on Artificial Neural Networks. Springer Berlin Heidelberg pp. 613–618 (1997)
16. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. Brazilian Symposium on Artificial Intelligence. Springer Berlin Heidelberg pp. 286–295 (2004)
17. Gama, J., Sebastiao, R., Rodrigues, P.: Issues in evaluation of stream learning algorithms. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining pp. 329–338 (2009)
18. Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Computing Surveys (CSUR) **46(4)**, 44 (2014)
19. Guenther, W.: Shortest confidence intervals. The American Statistician **23(1)**, 22–25 (1969)
20. Hong, S., Vatsavai, R.: Sliding window-based probabilistic change detection for remote-sensed images. Procedia Computer Science **80**, 2348–2352 (2016)
21. Jiang, D., Liu, J., Xu, Z., Qin, W.: Network traffic anomaly detection based on sliding window. IEEE International Conference on Electrical and Control Engineering pp. 4830–4833 (2011)
22. Jr, P.G., Barros, R.D.: Rcd: A recurring concept drift framework. Pattern Recognition Letters **34(9)**, 1018–1025 (2013)
23. Katakis, I., Tsoumakas, G., Vlahavas, I.: Tracking recurring contexts using ensemble classifiers: an application to email filtering. Knowledge and Information Systems **22(3)**, 371–391 (2010)
24. Kifer, D., Ben-David, S., Gehrke, J.: Detecting change in data streams. International Conference on Very Large Data Nases **30**, 180–191 (2004)
25. Kohonen, T.: The self-organizing map. Neurocomputing **21(1)**, 1–6 (1998)
26. Krawczyk, B., Wozniak, M.: One-class classifiers with incremental learning and forgetting for data streams with concept drift. Soft Computing **19(12)**, 3387–3400 (2015)
27. Li, K., Huang, H., Tian, S., Xu, W.: Improving one-class svm for anomaly detection. IEEE International Conference Machine Learning and Cybernetics **5**, 3077–3081 (2003)
28. Liu, F., Ting, K., Zhou, Z.: Isolation forest. IEEE International Conference on Data Mining pp. 413–422 (2008)
29. Losing, V., Hammer, B., Wersing, H.: Knn classifier with self adjusting memory for heterogeneous concept drift. International Conference On Data Mining (2016)
30. Marsland, S., Shapiro, J., Nehmzow, U.: A self-organising network that grows when required. Neural Networks **15(8)**, 1041–1058 (2002)

31. Martinetz, T.M., Berkovich, S.G., Schulten, K.J.: Neural-gas network for vector quantization and its application to time-series prediction. IEEE Transactions on Neural Networks **4(4)**, 558–569 (1993)
32. Masud, M., Chen, Q., Khan, L., Aggarwal, C., Gao, J., Han, J., Thuraisingham, B.: Addressing concept-evolution in concept-drifting data streams. IEEE International Conference on Data Mining pp. 929–934 (2010)
33. Middleton, B., Sittig, D., Wright, A.: Clinical decision support: a 25 year retrospective and a 25 year vision. Yearbook of Medical Informatics (2016)
34. Nishida, K., Shimada, S., Ishikawa, S., Yamauchi, K.: Detecting sudden concept drift with knowledge of human behavior. IEEE International Conference on Systems, Man and Cybernetics pp. 3261–3267 (2008)
35. Patel, H., Thakore, D.: Moving object tracking using kalman filter. International Journal of Computer Science and Mobile Computing **2(4)**, 326–332 (2013)
36. Pedregosa, F., G.Varoquaux, A.Gramfort, V.Michel, B.Thirion, O.Grisel, M.Blondel, P.Prettenhofer, R.Weiss, V.Dubourg, J.Vanderplas, A.Passos, D.Cournapeau, M.Brucher, M.Perrot, E.Duchesnay: Scikit-learn: machine learning in python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
37. Pentland, A., Liu, A.: Modeling and prediction of human behavior. Neural Computation **11(1)**, 229–242 (1999)
38. Prudent, Y., Ennaji, A.: An incremental growing neural gas learns topologies. IEEE International Joint Conference on Neural Networks **2**, 1211–1216 (2005)
39. Santosh, D., Venkatesh, P., Poornesh, P., Rao, L., Kumar, N.: Tracking multiple moving objects using gaussian mixture model. International Journal of Soft Computing and Engineering **3(2)**, 114–9 (2013)
40. Schlkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., Platt, J.: Support vector method for novelty detection. Advances in Neural Information Processing Systems pp. 582–588 (2000)
41. Shen, F., Q. Ouyang, W.K., Hasegawa, O.: A general associative memory based on self-organizing incremental neural network. Neurocomputing **104**, 57–71 (2013)
42. Souza, V., Silva, D., J. Gama, G.B.: Data stream classification guided by clustering on nonstationary environments and extreme verification latency. SIAM International Conference on Data Mining pp. 873–881 (2015)
43. Street, W., Kim, Y.: A streaming ensemble algorithm sea for large-scale classification. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining pp. 377–382 (2001)
44. Webb, G., Hyde, R., Cao, H., Nguyen, H., Petitjean, F.: Characterizing concept drift. Data Mining and Knowledge Discovery **30(4)**, 964–994 (2016)
45. Webb, G., Pazzani, M., Billsus, D.: Machine learning for user modeling. User Modeling and User-Adapted Interaction **11(1)**, 19–29 (2001)
46. Zeiler, M.: Adadelta: An adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)
47. Zhang, L., Lin, J., Karim, R.: Sliding window-based fault detection from high-dimensional data streams. IEEE Transactions on Systems, Man, and Cybernetics: Systems **47(2)**, 289–303 (2017)
48. Zliobaite, I., Pechenizkiy, M., Gama, J.: An overview of concept drift applications. In Big Data Analysis: New Algorithms for a New Society. Springer International Publishing pp. 91–114 (2016)