Exploiting the stimuli encoding scheme of evolving Spiking Neural Networks for stream learning

Jesus L. Lobo, Izaskun Oregi, Albert Bifet, Javier Del Ser

Exploiting a Stimuli Encoding Scheme of Spiking Neural Networks for Stream Learning

Jesus L. Lobo[a,*], Izaskun Oregi[a], Albert Bifet[b,c], Javier Del Ser[a,d,e]

[a]*TECNALIA, 48160 Derio, Spain.*
[b]*Télécom ParisTech, París, C201-2 France*
[c]*University of Waikato, Hamilton, New Zealand*
[d]*Basque Center for Applied Mathematics (BCAM), 48009 Bilbao, Spain*
[e]*University of the Basque Country UPV/EHU, 48013 Bilbao, Spain*

Abstract

Stream data processing has gained progressive momentum with the arriving of new stream applications and big data scenarios. One of the most promising techniques in stream learning is the Spiking Neural Network, and some of them use an interesting population encoding scheme to transform the incoming stimuli into spikes. This study sheds lights on the key issue of this encoding scheme, the Gaussian receptive fields, and focuses on applying them as a pre-processing technique to any dataset in order to gain representativeness, and to boost the predictive performance of the stream learning methods. Experiments with synthetic and real data sets are presented, and lead to confirm that our approach can be applied successfully as a general pre-processing technique in many real cases.

Keywords: Stream learning, gaussian receptive fields, population encoding, spiking neural networks

1. Introduction

The continuous production of tremendous amount of data in the form of fast streams upsets the traditional view in machine learning, thus giving rise to a new emerging paradigm called stream learning (SL). These streams of data evolve generally over time and may be occasionally affected by a change (concept drift) which impacts on their input data distribution, without following the fundamental hypothesis of stationarity upon which the learning theory is based. Learning in non-stationary environments has attracted much attention in the SL community in

*Corresponding author: jesus.lopez@tecnalia.com (Jesus L. Lobo). TECNALIA. P. Tecnologico Bizkaia, Ed. 700, 48160 Derio, Spain. Tl: +34 946 430 50. Fax: +34 901 760 009.

recent years due to its importance for many real-life applications, such as financial applications, network monitoring, cybersecurity, sensor networks, social networks analysis, among other Big Data scenarios (Chen, Mao & Liu, 2014). Learning under non-stationary conditions is especially challenging in Online Learning (OL) scenarios, where some systems may impose stringent restrictions, such as only a single sample is provided to the learning algorithm at every time instant, a very limited processing time, a finite amount of memory, and the necessity of having trained models at every scan of the streams of data.

The impact of SL and concept drift (CD) is particularly relevant in context of Internet of Things (IoT) (De Francisci Morales, Bifet, Khan, Gama & Fan, 2016), which sets forth a number of challenges that have to do with the nature of the data and the processes that generate them. Here the stationarity hypothesis is far from being the standard scenario: the data generation processes have a strong spatio-temporal dimension, which needs to be considered during the data modeling process if a SL algorithm claims to perform a reliable knowledge extraction. Moreover, IoT devices can regularly fail (e.g. limited battery life-time, loss of connectivity, failure, aging, overheating, etc.) resulting in a change that may affect data distribution. These changes causes that, predictive models trained over these IoT stream data before the change occurs, become obsolete, and do not adapt suitably to the new emerging distribution. Any learning algorithm that is going to be used in such a scenario should be able to cope with CD in evolving environments.

Many of the traditional ML algorithms need to be retrained if they are used in a changing environment, and they fail to scale properly. For this reason there is a pressing need for new algorithms that adapt to changes as fast as possible, while providing good performance scores. A large number of algorithms have been developed to deal eather with CD adaptation (Gama, Žliobaitė, Bifet, Pechenizkiy & Bouchachia, 2014; Webb, Hyde, Cao, Nguyen & Petitjean, 2016) and/or CD detection (Barros & Santos, 2018). Some of these adaptation techniques rely on artificial neural networks (ANNs), such as Multilayer Perceptron (Minku & Yao, 2012; Polikar, Upda, Upda & Honavar, 2001), which are a biologically inspired paradigm that mimics the process that brain acquires and processes sensory information. Considered as the third generation of ANNs, Spiking Neural Networks (SNNs) are one of the most biologically plausible approaches (Gerstner & Kistler, 2002; Kasabov, 2018) thanks to their neuron model and their realistic brain-like information processing, which eases their implementation on super-fast and reliable hardware architectures. Especially in SL, some SNNs (e.g. evolving SNNs) are found as a reputed approach for their ability to learn continuously and incrementally, which account for their adaptability to non-stationary and evolving

scenarios. It is known that brains do not deal with real numbers but with timed spikes, and a key aspect of the SNNs architecture is how information is encoded with such spikes. Gaussian Receptive Fields (GRFs) population encoding scheme constitutes a biologically plausible and well studied method for representing real-valued parameters (Bohte, Kok & La Poutre, 2002). However, there is a lack of research on how the application of a GRFs population encoding scheme to a dataset can improve by itself the predictive performance of a dataset, without being a mere encoding module of a SNN, and becoming a relevant pre-processing technique for the SL field. It is here where we find an interesting research gap and challenge to be tackled in this study.

Data pre-processing (DP) has become essential in current knowledge discovery scenarios (dominated by increasingly large datasets like in SL), which aims at getting more precise learning process, among others goals (Ramírez-Gallego, Krawczyk, García, Woźniak & Herrera, 2017). Concretely, space transformations generate a whole new set of features by combining or transforming the original ones. Most of the space transformation approaches focus on reducing the number of dimensions. However, this study proposes to increase them by applying a GRFs population encoding scheme, which will turn into a higher representativeness of the input data, and a predictive performance improvement balanced with the amount of time spent on the computations. As recent studies of DP (García, Ramírez-Gallego, Luengo, Benítez & Herrera, 2016; Ramírez-Gallego, Krawczyk, García, Woźniak & Herrera, 2017) encourage researchers to develop new DP techniques for data streams, we have decided to consider it as a general pre-processing technique that can be applied to any SL method of the literature. This approach will be tested with many of the current SL methods and datasets in the state of the art, as we will show in next sections.

The study is organized as follows: first, Section 2 provides a general introduction to the GRFs population encoding scheme in SNNs. Section 3 presents an insight about the impact of the GRFs population encoding parameters on the stream data representation. Section 4 delves into a detailed description of the proposed approach, while Section 5 presents the experiments. Sections 6 and 7 show and discuss the obtained results from such experiments respectively. And finally, Section 8 draws conclusions and proposes future research lines related to this study.

2. Spiking Neural Networks and Stimuli Encoding Schemes

SNNs have recently attracted much attention in OL due to: 1) their ability to capture temporal dependence of stream data, 2) they are able to learn continuously

and incrementally, 3) they do not need to be retrained in a fast evolving environment, and 4) because they are trained innately in an online manner. Besides, they mimic the process through which the brain acquires and processes sensory information thanks to their biologically plausible neuron models. The use of SNNs in OL allows for a very fast real-time and reducing the computational complexity of the learning process, and this is the case of some approaches like SpikeProp (Bohte, Kok & La Poutre, 2002), ReSuMe (Ponulak, 2005), SpikeTemp (Wang, Belatreche, Maguire & McGinnity, 2017), or the recent work presented in (Lobo, Laña, Del Ser, Bilbao & Kasabov, 2018b) where the evolving SNNs were modified to fit the OL requirements in a more realistic way. They have been used even as drift detectors in (Lobo, Del Ser, Laña, Bilbao & Kasabov, 2018a). Evolving SNNs are a successful type of SNN (Schliebs & Kasabov, 2013), where the number of spiking neurons evolves incrementally in time to infer temporal patterns from data. Precisely one of the key ingredient of evolving SNNs is their temporal encoding module (see Figure 1). The traditional form of patterns which usually consists of real values cannot be used to feed the SNN in a simulation process, and they need to be transformed into temporal patterns (such as events in time or spike trains). Before presenting this pattern to the network, real values of the features of every sample are encoded into spike trains, being a process that aims at generating a new representation of the input stimuli in more dimensions.

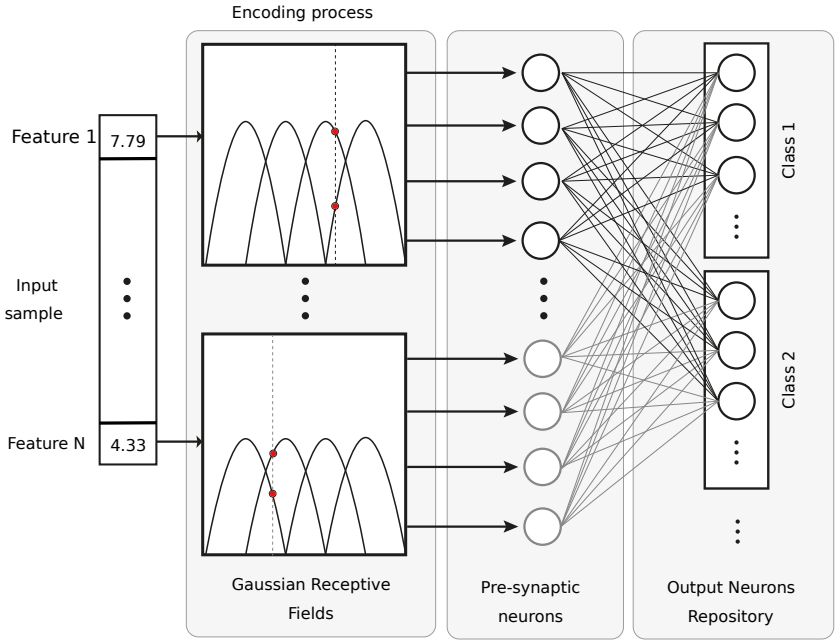


Figure 1: Architecture of an evolving SNN (Kasabov, 2007).

SNNs are composed by three layers (see Figure 1): the first one corresponds to the input data (stimuli). The second layer addresses the encoding process, where the real values of the features of every sample are encoded as trains of spikes by using GRFs. The third layer is the evolving output layer, being a repository of spiking neurons that represent samples and their class labels; they evolve as new samples arrive. Each output neuron is linked to all input neurons through connections whose weights are learned from the samples fed to the network. The second layer is precisely where we focus our encoding approach.

Rank Order Population encoding (Bohte, Kok & La Poutre, 2002) is used in the second layer, and it is an extension of the Rank Order encoding introduced in (Thorpe & Gautrais, 1998). Basically, it allows the mapping of vectors of real values into a sequence of spikes, and it is based on receptive fields which encode the real values by using a collection of Gaussian curves with overlapping sensitivity profiles. Each feature is encoded independently by a number of receptive fields (*n_GRFs*). GRFs overlap with each other by adopting the shape of a Gaussian function, in all cases covering the whole range of the values of each feature. The parameter *n_GRFs* may vary depending on the nature of the data at hand, and must be tuned for achieving a good predictive performance of the overall model. Concretely, the center C_i and the width W_i of each GRF i of the feature f are computed as

$$C_i = I^n_{min} + \frac{2j-3}{2}\left(\frac{I^n_{max} - I^n_{min}}{n\_GRFs - 2}\right) \tag{1}$$

and

$$W_i = \frac{1}{\gamma}\left(\frac{I^n_{max} - I^n_{min}}{n\_GRFs - 2}\right), \tag{2}$$

where n_GRFs is the number of Gaussian receptive fields (equally spaced Gaussian curves), whose value impacts on the amplitude of the input neuron. The range of the n-th input feature is $\mathbb{R}[I^n_{min}, I^n_{max}]$. Parameter γ (overlap factor) regulates the width of the GRFs, thereby their amount of overlapping. Each feature will be transformed in a real values vector (spikes train), defined as

$$vector_f = \exp\left(-\frac{(x - C_i)^2}{2W_i^2}\right), \tag{3}$$

where x is the input value of the feature f. Figure 2 exemplifies the GRFs encoding process for one of the features (0.73) of any given input sample.

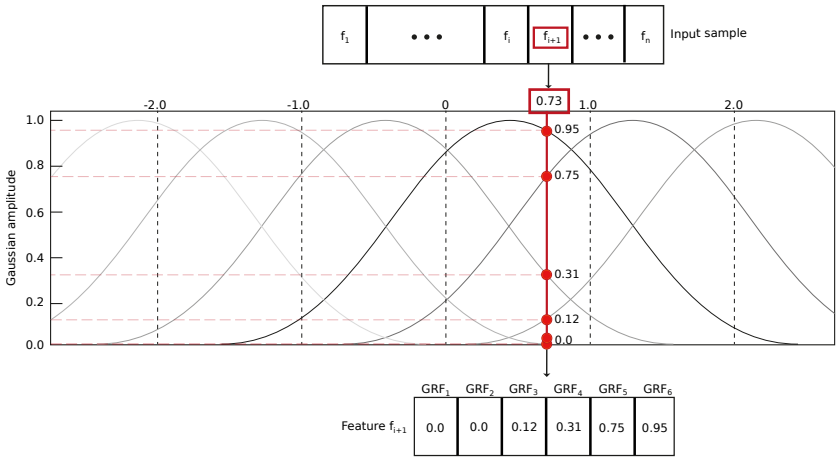


Figure 2: Example of GRFs population encoding based on 6 Gaussians. For an input value of the feature f_{i+1} (0.73, bold straight line), the intersection points with each GRF are computed $(0.95, 0.75, 0.31, 0.12, 0.0, 0.0)$, which are in turn translated into a vector of real values for the feature f_{i+1}.

3. Impact of the GRFs Parameters on Stimuli Representation

Before presenting our approach in the next Section, we would like to introduce some insights of the GRFs population encoding scheme when applied to stream data, by showing the impact of the GRFs parameters on the resulting vector of real values. Firstly, and as explained in Section 2, the result of the encoding process of a dataset is clearly depicted in Figure 3, where it is shown how the parameter *n_GRFs* impacts on the number of final encoded real values. The more GRFs are used the more cut points with the Gaussian curves are present, and the more real encoded values will be (see lines 11 and 16 in Algorithm 1, and Figure 2), which will impact directly in the time processing of each sample, as we will see in Section 7.

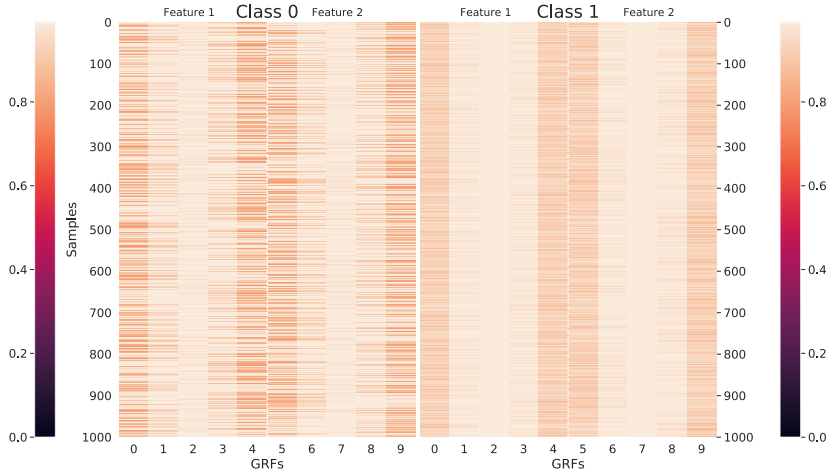
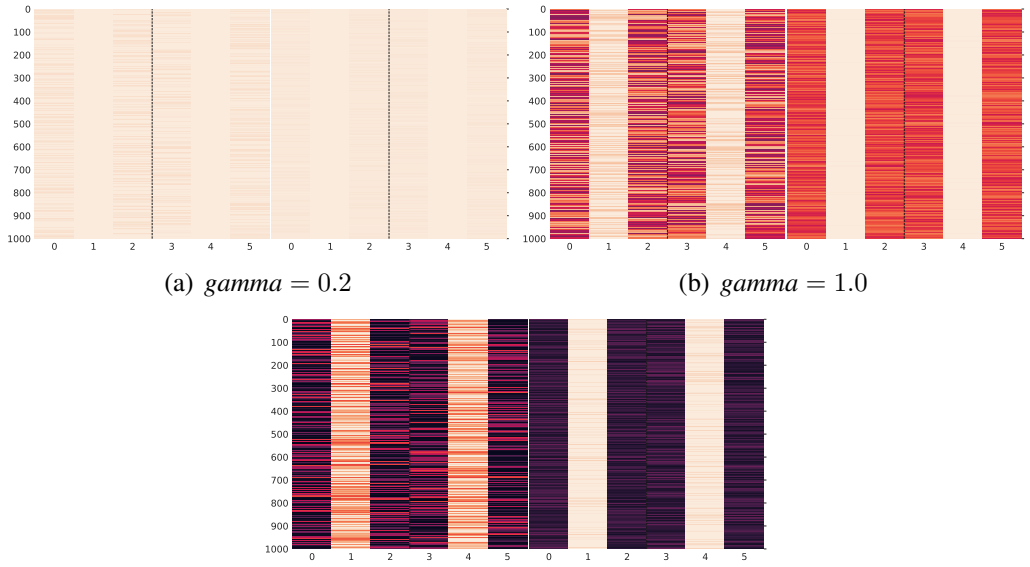


Figure 3: Example of GRFs population encoding: this binary class dataset with 1000 samples and 2 features has been encoded with 5 GRFs (per feature), resulting in an encoded vector of 10 real values for the samples belonging to the class 0, and also an encoded vector of 10 real values for the samples belonging to the class 1. The color maps correspond to the range of values of the real encoded values.

In Figure 4 we can see the impact of the parameter *gamma* on the resulting vector of real values that represent the sample after the GRFs population encoding. The higher value of *gamma* is the lower encoded values are obtained. This makes sense when we see in Figure 2 that the overlapping area between Gaussian curves is defined by *gamma* parameter; with a high value of *gamma* we obtain less overlapping and thus the encoded real values will be closer to 0 (the lower limit of the transformation interval) more frequently. However, with a low value of *gamma*, the overlapping area between Gaussian curves is higher, what provokes that the values of the cut points are closer to the upper limit of the transformation interval. Therefore, when the value of *gamma* is too high or too low (there is no so variety in the values representation, all of them are close to the lower or upper limit of the transformation interval respectively), and then the final encoded vector loses representativeness. The increase in representativeness is precisely the goal of using a GRFs population encoding scheme in this study; by augmenting the representativeness of input data we may increment the predictive power of stream learning methods in many cases, as it will be shown in Sections 6 and 7. It is necessary to find the suitable level of overlapping to maximize this predictive performance gain.

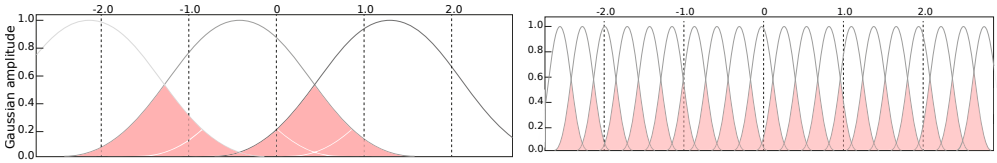


(a) *gamma* $= 0.2$ (b) *gamma* $= 1.0$

(c) *gamma* $= 2.0$

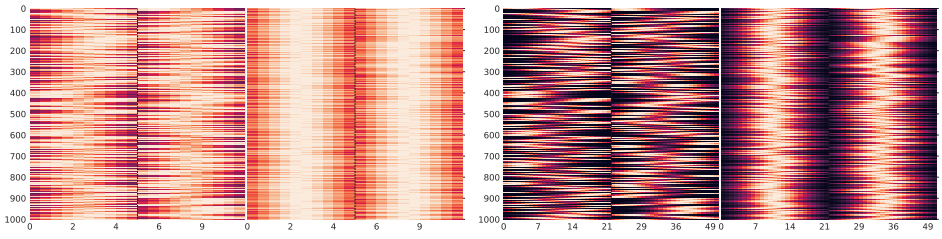
Figure 4: Impact of the different values of *gamma* on the real encoded values. They represent an overlap of 2%, 10%, and 20% between two subsequent GRFs respectively. The parameter *n_GRFs* is 3. The characteristics of the dataset and the color map are the same than in Figure 3.

The same situation may happen with *n_GRFs* parameter, as it is reflected by Figures 6 and 7. With a high number of GRFs we obtain a higher number of cut points (see Figure 2) and then a bigger vector of real encoded values. As we see in Figures 6 and 7, by fixing the *gamma* parameter and varying the *n_GRFs* parameter, we also get different levels of representativeness, which lead us to think that there is a trade-off between both parameters of the encoding scheme (see Figure 5). In fact, by combining the suitable values of both parameters we can achieve a similar dataset representation (Figure 6c and 7a).

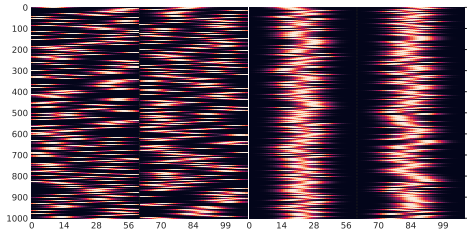


(a) 3 GRFs with a low value of *gamma* (b) 20 GRFs with a high value of *gamma*

Figure 5: Achieving similar overlapping by controlling *gamma* and *n_GRFs*.

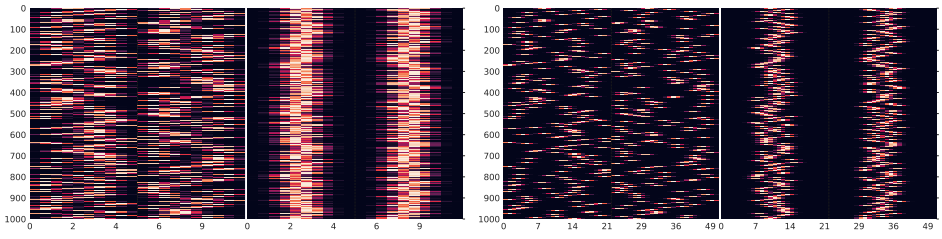


(a) $n\_GRFs = 10$ (b) $n\_GRFs = 25$

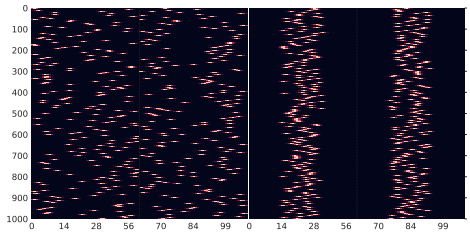


(c) $n\_GRFs = 50$

Figure 6: Impact of the different values of *n_GRFs* per feature on the real encoded values. The parameter *gamma* is 0.2. The characteristics of the dataset and the color map are the same than in Figure 3.



(a) $n\_GRFs = 10$ (b) $n\_GRFs = 25$



(c) $n\_GRFs = 50$

Figure 7: Impact of the different values of *n_GRFs* per feature on the real encoded values. The parameter *gamma* is 1.0. The characteristics of the dataset and the color map are the same than in Figure 3.

Finally, it should be highlighted the fact that features may have different data distributions and representations for each class, and thus the choice of *gamma* and *n_GRFs* values may affect the features in a different way. This effect can be corroborated in Figures 3, 4, 6, and 7.

The goal of this study is to look into the benefits of gaining representativeness of the data, by focusing on its impact on the predictive performance of the stream learning methods. In the next Section we explain our approach from a practical view in stream learning, and after that we set up the experiments to confirm our assumptions.

4. Proposed Approach

Our proposed approach consists of applying GRFs population encoding scheme to any SL method, with the objective of achieving an improvement of its predictive performance. This encoding scheme is already used in the literature, but only as a mere encoding module of the evolving SNNs, and there is a lack of research on how its application to any stream learner as a DP technique can be the key to obtain a better predictive performance.

The application of a GRFs population encoding scheme to any streaming learner is described in Algorithm 1. The aim of this process is to transform the original feature space of a sample (line 6) into a new set of real values (*vector_s* of line 7). The transformation process of each feature (line 8) is based on the creation of a number (*n_GRFs*) of GRFs (with their own width and center in lines 9 and 12 respectively) and their cut points (line 13) between the GRFs with the real value that represents each feature (see Figure 2). At the end of the process (line 17), the sample will be represented by a new set of real values (*vector_s*). Once the sample has been transformed, a *test-then-train* scheme (see subsection 5.5) is applied, where each sample is firstly used for testing the model before it is used for training (lines 18 and 24). During this process, and after the drift detector has been updated with the stream learner prediction (line 19), a drift may appear (line 20), and then the drift detector and the stream learner need to be initialized (lines 21 and 22).

Algorithm 1: Proposed approach for applying GRFs to any SL method

```
Set GRFs population encoding parameters: α, n_GRFs
Stream_learner=[KNN, HT, HAT, MNB, GNB, SGD, Perceptron, PA,
  MLP] (See subsection 5.3)
Drift_detector=ADWIN() (See subsection 5.4)
Calculate I_max, I_min with a portion of the stream data
Perform a warm start of the stream_learner with the same portion of the
  stream data
for every sample s in the rest of the stream data do
    Initialize vector_s
    for every feature f do
        Calculate W_f
        Initialize vector_f
        for every GRF in n_GRFs f do
            Calculate C_f
            Calculate cut points for the GRF
            Concatenate cut points to vector_f
        end
        Concatenate vector_f to vector_s
    end
    Test stream_learner with vector_s
    Update Drift_detector with stream_learner's prediction
    if Drift_detector == True then
        Initialize Stream_learner
        Initialize Drift_detector
    end
    Train stream_learner with vector_s
end
```

5. Experiments Design

An extensive experimental benchmark has been designed in order to confirm the feasibility of applying GRFs population encoding scheme to several SL methods. To achieve this goal, we have selected some of the most utilized stream learners in the literature, and have modified them to include the GRFs population encoding scheme. Finally, we have tested them against their original versions in several synthetic and real datasets following the standards of the SL evaluation. The experiments are composed of 9 pairs of techniques (original technique vs original technique with GRF population encoding scheme), which are compared

in terms of predictive performance and time processing, with a statistical significance test in order to know about the improvement of applying GRFs population encoding scheme to the original versions of the SL methods. Every experiment has been carried out 25 times. In the next subsections we will detail the set up of these experiments.

5.1. Framework

The experiments have been carried out under the *scikit-multiflow* framework (Montiel, Read, Bifet & Abdessalem, 2018), which is implemented in Python given its increasing popularity in the ML community. It is inspired by the most popular open source Java framework for data stream mining, MOA (Bifet, Gavaldà, Holmes & Pfahringer, 2018), and it includes a collection of ML algorithms (classification, regression, clustering, outlier detection, CD detection and recommender systems), datasets, and tools and metrics for SL evaluation. It complements *scikit-learn*[1], whose primary focus is batch learning (despite the fact that it also provides researchers with some OL methods) and expands the set of ML tools on this platform.

5.2. Datasets

The proposed approach is tested with synthetic and real datasets. On the one hand, synthetic datasets are easier to reproduce and identify the data distribution (concept). On the other hand, real-world data allow us to test our approach under real conditions, but without knowing anything about the existence of drifts, their nature (severity, velocity, recurrence, etc.) in case of being present, or the data distribution.

We have selected four well-known synthetic datasets (Minku, White & Yao, 2010): `CIRCLE`, `LINE`, `SINEH` and `SINEV`. Each dataset consists of $2,000$ samples, 2 normalized and continuous features $\{X_1, X_2\}$, and represents a binary problem. Drift appears at $t = 1,000$ in all of them, and we have divided each dataset into 2 different datasets: one before the drift (first concept) with $1,000$ samples and other one after the drift (second concept) with $1,000$ samples. After that, we have replicated each dataset 50 times in order to obtain larger datasets more appropriately to serve as test data in a SL process, resulting the following datasets: *circle_concept1*, *circle_concept2*, *line_concept1*, *line_concept2*, *sine_concept1*, *sine_concept2*, *sineH_concept1*, and *sineH_concept2*. Resulting ultimately in 8 different synthetic datasets of $50,000$ samples. In the case of real-world scenarios, we have resorted to 5 different datasets:

[1] https://scikit-learn.org/stable/

- *Weather* dataset (Elwell & Polikar, 2011). The U.S. National Oceanic and Atmospheric Administration has compiled a database with $18,159$ daily weather measurements (50 years) from over $9,000$ weather stations all around the world. Data samples are composed of 8 features (temperature, dew point, sea level pressure, visibility, average wind speed, and other weather related predictors alike). The goal is to infer whether each day was rainy or not. It is available in *scikit-multiflow*.
- *Electricity market* dataset (Harries & Wales, 1999). The dataset is based on $45,312$ instances dated from May 1996 to December 1998. Each sample refers to a period of 30 minutes, and has 5 features (day of week, time stamp, New South Wales electricity demand, Victoria electricity demand, and scheduled electricity transfer between states). The dataset corresponds to a binary problem, and the target identifies the change of the price (up or down) related to a moving average of the last 24 hours. It is available in *scikit-multiflow*.
- *Moving squares* dataset (Losing, Hammer & Wersing, 2016). 4 equidistantly separated, squared uniform distributions are moving horizontally with constant speed. The direction is inverted whenever the leading square reaches a predefined boundary. Each square represents a different class. The added value of this dataset is the predefined window of 120 samples before old samples may start to overlap current ones. The dataset contains $200,000$ samples and corresponds to a multi-class classification problem. It is available in *scikit-multiflow*.
- *SEA* dataset (Street & Kim, 2001). It consists of 3 numerical attributes that vary from 0 to 10, where only 2 of them are relevant to the classification task. A classification function is chosen, among four possible ones. These functions compare the sum of the two relevant attributes with a threshold value, unique for each of the classification functions. Depending on the comparison, the generator will classify an instance as one of the two possible labels. It is available in *scikit-multiflow*, and it has been generated with its *SEAGenerator* method.
- *Airlines* dataset (Ikonomovska, Gama & Džeroski, 2011). It contains $539,383$ examples described by 7 features (3 numeric and 4 nominal). Airlines encapsulates the binary task of predicting whether a given flight will be delayed, given the information of the scheduled departure.

For the sake of time efficiency in the experiments, we have limited the stream data size to the first $50,000$ samples in the case of *Moving Squares* and *Airlines* datasets. Finally, it should be mentioned that synthetic datasets are balanced, whereas real ones are imbalanced data, which will determine the choice of the accuracy performance metric in subsection 5.5.

5.3. Stream Learning Methods

The SL methods of this study have been chosen due to their wide use in the state of the art; in fact, all of them are available in the *scikit-multiflow* and *scikit-learn* frameworks. The stream learners are:

- *K-Nearest Neighbors* classifier (Dasarathy, 1991), labeled as `KNN`. It works by keeping track of a fixed number (the last max_window_size) of training samples. Then, it searches its stored samples and find the closest ones using a selected distance metric.
- *Hoeffding Tree* or *Very Fast Decision Tree* (Hulten, Spencer & Domingos, 2001), labeled as `HT`. It is an incremental decision tree induction algorithm capable of learning from massive data streams, and assumes that the distribution generating samples does not change over time. `HT` exploits the fact that a small sample can often be enough to choose an optimal splitting attribute. This idea is supported mathematically by the Hoeffding bound, which quantifies the number of samples needed to estimate some statistics within a prescribed precision (e.g. the goodness of an attribute).
- *Hoeffding Adaptive Tree* (Bifet & Gavaldà, 2009), labeled as `HAT`. It uses a drift detector, `ADWIN` (Bifet & Gavalda, 2007), to monitor performance of branches on the tree and to replace them with new branches when their accuracy decreases if the new branches are more accurate.
- *Multinomial Naive Bayes*, labeled as `MNB`. It implements the Naive Bayes algorithm (Zhang, 2004) for multinomially distributed data.
- *Gaussian Naive Bayes* (Chan, Golub & LeVeque, 1982), labeled as `GNB`. It updates means and variances in an online manner.
- *Stochastic Gradient Descent* classifier (Robbins & Monro, 1985), labeled as `SGD`. It implements regularized linear models with stochastic gradient descent learning. The gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (learning rate).
- *Perceptron* (Freund & Schapire, 1999). It is a classification algorithm very similar to `SGD` classifier. In fact, they share the same underlying implementation, but *Perceptron* uses the perceptron loss function instead of hinge function. By default, *Perceptron* does not require a learning rate, it is not regularized, and it is updated only on mistakes. These characteristics imply that *Perceptron* is slightly faster to train than `SGD` and that the resulting model is sparser.
- *Passive Agressive* classifier (Crammer, Dekel, Keshet, Shalev-Shwartz & Singer, 2006), labeled as `PA`. It is similar to *Perceptron* in that it does not require a learning rate. However, it includes a regularization parameter *C*.

- *Multi-layer Perceptron* classifier (Hinton, 1990), labeled as `MLP`. It trains iteratively since at each time step the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters.

`KNN`, `HT`, `HAT` and `MNB` are available in *scikit-multiflow*, while `GNB`, `SDG`, `Perceptron`, `PA`, and `MLP` are available in *scikit-learn* between its options for incremental learning and strategies to scale computationally. Finally, it should be highlighted that all stream learners have been pre-trained with a number of samples before starting the evaluation, in order to enforce a warm start. These pre-training samples are also used to calculate I_{max}, I_{min}, and these limits will be used for the rest of the stream data. Other valid option is not to use this pre-training, and update I_{max} and I_{min} values every time a new sample is available.

5.4. Drift Detection and Adaptation Mechanisms

Although the CD detection and adaptation is not the core of this study, it is present in many real streaming processes. For the real-world experiments the drift appearance is unknown, and then a drift detector is needed to identify it as soon as possible, and to trigger an adaptation mechanism (in this work we have opted for an active approach). `ADWIN` (Bifet & Gavalda, 2007) is a drift detector which maintains a window of variable size containing samples, and automatically grows the window size when no change is apparent, and shrinks it when data changes. It has been selected because can work together with any learning algorithm, it requires low time and memory resources, and provides rigorous guarantees of its performance in the form of limits on the rates of false positives and false negatives (Gonçalves Jr, de Carvalho Santos, Barros & Vieira, 2014).

Regarding the adaptation, as we do not know anything about the nature of the drifts in the real cases, we can not opt for a suitable adaptation technique beforehand, and we have initialized the learners after drift is detected. Besides, as we have carried out an OL approach in which only one sample is available at each moment, there is no option for storing a window of past samples and performing a forgetting mechanism based on windowing (or other sample storing scheme). The impact of the drift detection and adaptation mechanisms are not the core of this study, and we encourage other researchers to consider other mechanisms.

5.5. Streaming Evaluation Methodology

Evaluation is a fundamental task to know when an approach is outperforming another method only by chance, or when there is a statistical significance to that claim. In the case of SL, the methodology is very specific to consider the fact that not all data can be stored in memory (e.g. in OL only one sample is processed at each time), and that data can follow a non-stationary distribution (drifts may occur

at any time). We have followed the evaluation methodology proposed in (Gama, Žliobaitė, Bifet, Pechenizkiy & Bouchachia, 2014; Bifet, de Francisci Morales, Read, Holmes & Pfahringer, 2015; Bifet, Gavaldà, Holmes & Pfahringer, 2018), and that recommends to follow these guidelines in several evaluation tasks:

- *Error estimation*. We have used an *interleaved test-then-train* scheme, where each sample is firstly used for testing the model before it is used for training, and from this, the accuracy metric is incrementally updated. The model is thus always being tested on samples it has not seen. In our study we have not used a landmark window, and only one sample is used at each time step.
- *Performance evaluation measure*. As in real data streams the number of samples for each class may be evolving and changing, we have opted for the Kappa statistic (*K*) (Cohen, 1960) because it is a more sensitive measure for quantifying the predictive performance of streaming classifiers:

$$K = \frac{p_o - p_c}{1 - p_c} \tag{4}$$

  , where p_o is the prequential accuracy of the classifier, and p_c is the probability that a chance classifierone that randomly assigns to each class the same number of samples as the classifier under considerationmakes a correct prediction.
- *Statistical significance*. When comparing two classifiers, it is necessary to distinguish whether a classifier is better than another one only by chance, or whether there is a statistical significance to ensure that. The McNemar test (McNemar, 1947) is a non-parametric test used in SL to assess the differences in the performance of two classifiers. The McNemar statistic (*M*) is given as

$$M = \frac{(a-b)^2}{a+b} \tag{5}$$

  , where *a* is the number of samples misclassified by the first classifier and correctly classified by the second one, and *b* is the number of samples misclassified by the second classifier and correctly classified by the first one. The test follows the $\tilde{\chi}^2$ distribution. At 0.95 confidence it rejects the null hypothesis if $M > 3.841459$ (Dietterich, 1998). Because it is well known that the probability of signaling differences where they do not exist is highly affected by data length, we have computed the McNemar test over a sliding window of 500 samples. (Gama, Sebastião & Rodrigues, 2013). The null hypothesis states that augmenting the representativeness of the incoming data by applying a GRFs population encoding scheme, the predictive performance of the stream learning algorithms will be the same than without applying this scheme. We have

to reject the null hypothesis at least more than 50% during the SL process to consider that our approach starts to show significance.

- *Performance benchmarking*. As it has been shown in subsection 5.2, we have taken into consideration both large synthetic and real datasets.
- A *cost measure*. We have opted for measuring the processing time (in seconds) of the stream learner in each dataset. The computer that has carried out the experiments is based on a x86_64 architecture with 8 processors Intel(R) Core(TM) i7 at 2.70GHz, and 32 DDR4 memory running at $2,133$ MHz.

5.6. Parameters Choice

Because the best performance of the stream learners is not the goal of this study, we have not carried out a parameter tuning task thoroughly. We are interested in showing the improvement of the classification performance and its statistical significance when GRFs population encoding is applied to the benchmarking. The most relevant parameters configuration is shown in Tables A.9, A.10, and A.11 of the Appendix A.

6. Results

In this section we present the results of the SL methods when the GRF population encoding scheme is applied as a pre-processing technique to synthetic and real-world datasets. We evaluate the classification performance with the Kappa statistic, and the SL processing time in seconds. Regarding the statistical significance, the McNemar column shows the percentage of the data stream in which the null hypothesis is rejected. The null hypothesis to be tested is that there are no significant differences in terms of predictive performance between the original stream learners and those which use the DP technique based on the application of the GRFs encoding scheme.

Tables 1, 2, 3, and 4 collect the results for the synthetic dataset, whereas Tables 5, 6, and 7 present the results for the real datasets. Finally, Table 8 summarizes the results in order to have a clearer view of the impact of our approach for each SL method (Bifet, Holmes, Pfahringer & Frank, 2010), providing the mean of the results for all datasets.

Stream Learners	Kappa		McNemar		Time	
	circle_concept_1	circle_concept_2	circle_concept_1	circle_concept_2	circle_concept_1	circle_concept_2
KNN	0.362	0.300	66.59%	100.00%	12.82 ± 0.97	13.02 ± 2.35
GRF_KNN	0.414	0.418			12.58 ± 1.10	13.21 ± 2.25
HT	0.980	0.959	56.89%	100.00%	6.03 ± 0.50	7.02 ± 2.71
GRF_HT	0.998	0.987			8.67 ± 0.28	9.27 ± 2.31
HAT	0.924	0.881	14.79%	98.55%	12.10 ± 0.42	12.58 ± 1.35
GRF_HAT	0.932	0.886			16.30 ± 0.51	16.64 ± 1.97
MNB	-0.240	-0.182	100.00%	100.00%	17.23 ± 1.01	17.60 ± 2.35
GRF_MNB	0.794	0.868			16.62 ± 0.69	17.15 ± 2.16
GNB	0.964	0.946	0.00%	100.00%	14.14 ± 0.68	14.60 ± 1.84
GRF_GNB	0.966	0.950			15.59 ± 0.71	15.88 ± 2.02
SGD	0.012 ± 0.003	-0.104 ± 0.005	100.00%	100.00%	11.09 ± 0.61	11.22 ± 1.42
GRF_SGD	0.923 ± 0.001	0.929 ± 0.001			11.77 ± 0.50	12.12 ± 1.67
Perceptron	-0.679 ± 0.004	-0.691 ± 0.002	100.00%	100.00%	8.54 ± 0.37	8.79 ± 1.02
GRF_Perceptron	0.981 ± 0.007	0.931 ± 0.008			10.67 ± 0.50	11.80 ± 3.71
PA	-0.528 ± 0.003	-0.668 ± 0.001	100.00%	100.00%	8.44 ± 0.37	8.74 ± 1.01
GRF_PA	0.925	0.926			10.59 ± 0.48	11.14 ± 1.95
MLP	0.949 ± 0.002	0.952 ± 0.003	36.09%	93.90%	34.72 ± 1.73	35.48 ± 4.36
GRF_MLP	0.965 ± 0.001	0.962 ± 0.001			37.68 ± 1.55	9.32 ± 1.47

Table 1: Results for synthetic datasets *circle_concept1* and *circle_concept2*. Kappa statistic column evaluates the classification performance, McNemar column shows the percentage of the data stream in which the null hypothesis is rejected, and Time column reflects the duration of the SL process in seconds. The proposed approaches in the column Stream Learners are in italics.

Stream Learners	Kappa		McNemar		Time	
	line_concept_1	line_concept_2	line_concept_1	line_concept_2	line_concept_1	line_concept_2
KNN	0.756	0.778	92.59%	100.00%	10.36 ± 0.10	15.03 ± 1.98
GRF_KNN	0.644	0.718			10.79 ± 0.07	15.13 ± 1.85
HT	0.997	0.997	0.00%	0.34%	4.67 ± 0.03	6.39 ± 0.94
GRF_HT	0.997	0.986			7.51 ± 0.05	9.17 ± 1.32
HAT	0.978	0.970	30.19%	31.27%	11.78 ± 0.07	14.80 ± 4.43
GRF_HAT	0.948	0.953			16.06 ± 2.40	19.15 ± 3.83
MNB	0.594	0.578	0.00%	100.00%	15.36 ± 0.16	19.32 ± 3.38
GRF_MNB	0.974	0.903			15.50 ± 0.79	19.49 ± 4.64
GNB	0.964	0.948	0.00%	0.00%	12.97 ± 0.11	16.20 ± 2.74
GRF_GNB	0.962	0.956			14.30 ± 0.50	17.69 ± 3.39
SGD	0.972	0.969	3.76%	4.45%	9.78 ± 0.09	12.23 ± 2.12
GRF_SGD	0.953 ± 0.002	0.952 ± 0.001			10.85 ± 0.10	13.13 ± 2.36
Perceptron	0.976 ± 0.011	0.978 ± 0.012	73.65%	89.30%	7.88 ± 0.07	9.46 ± 1.69
GRF_Perceptron	0.954 ± 0.018	0.940 ± 0.003			9.84 ± 0.08	11.88 ± 2.11
PA	0.969 ± 0.001	0.970	3.88%	42.80%	7.97 ± 0.07	9.29 ± 1.68
GRF_PA	0.959	0.958			9.94 ± 0.08	11.77 ± 2.09
MLP	0.988	0.984	11.00%	57.45%	31.96 ± 0.37	38.28 ± 6.53
GRF_MLP	0.983 ± 0.001	0.985			34.83 ± 0.43	42.78 ± 9.70

Table 2: Results for synthetic datasets *line_concept1* and *line_concept2*.

Stream Learners	Kappa		McNemar		Time	
	sine_concept_1	**sine_concept_2**	**sine_concept_1**	**sine_concept_2**	**sine_concept_1**	**sine_concept_2**
KNN	-0.015	-0.009	1.00%	13.92%	13.93 ± 0.10	13.28 ± 0.71
GRF_KNN	-0.025	-0.018			14.17 ± 0.09	13.52 ± 0.92
HT	0.152	0.139	100.00%	32.99%	6.19 ± 0.07	6.21 ± 0.25
GRF_HT	0.173	0.160			9.41 ± 0.11	9.50 ± 0.14
HAT	-0.710	-0.149	60.37%	66.43%	11.21 ± 0.16	11.64 ± 0.09
GRF_HAT	-0.044	0.037			16.94 ± 0.20	16.77 ± 0.17
MNB	0.013	0.001	100.00%	0.00%	17.36 ± 0.33	16.93 ± 0.35
GRF_MNB	0.042	0.023			17.11 ± 0.18	16.47 ± 0.31
GNB	0.038	0.022	3.10%	37.29%	14.78 ± 0.18	14.16 ± 0.32
GRF_GNB	0.033	0.044			16.06 ± 0.25	15.39 ± 0.30
SGD	-0.231 ± 0.006	-0.300 ± 0.006	94.54%	92.71%	11.14 ± 0.22	10.65 ± 0.21
GRF_SGD	-0.291 ± 0.009	-0.298 ± 0.006			11.89 ± 0.21	11.38 ± 0.16
Perceptron	-0.696 ± 0.001	-0.781 ± 0.003	7.24%	25.62%	8.64 ± 0.14	8.29 ± 0.11
GRF_Perceptron	-0.731 ± 0.011	-0.737 ± 0.038			10.77 ± 0.18	10.39 ± 0.14
PA	0.626 ± 0.003	-0.658 ± 0.001	64.29%	57.12 %	8.46 ± 0.14	8.33 ± 0.09
GRF_PA	-0.881 ± 0.001	-0.915 ± 0.001			10.65 ± 0.17	10.44 ± 0.15
MLP	0.040 ± 0.004	-0.033 ± 0.005	26.50%	65.09%	35.02 ± 0.59	34.05 ± 0.43
GRF_MLP	0.050 ± 0.006	0.049 ± 0.011			38.70 ± 0.46	37.65 ± 0.50

Table 3: Results for synthetic datasets *sine_concept1* and *sine_concept2*.

Stream Learners	Kappa		McNemar		Time	
	sineH_concept_1	**sineH_concept_2**	**sineH_concept_1**	**sineH_concept_2**	**sineH_concept_1**	**sineH_concept_2**
KNN	0.329	0.270	0.00%	11.38%	10.54 ± 0.03	12.80 ± 1.49
GRF_KNN	0.304	0.288			10.90 ± 0.06	13.09 ± 1.42
HT	0.685	0.657	1.69%	1.56%	5.19 ± 0.02	5.87 ± 0.45
GRF_HT	0.845	0.808			8.93 ± 0.07	9.26 ± 0.21
HAT	0.426	0.452	100.00%	94.40%	11.81 ± 0.05	12.27 ± 0.21
GRF_HAT	0.530	0.539			16.01 ± 0.09	16.98 ± 0.47
MNB	0.053	0.122	49.47%	12.76%	15.96 ± 0.02	16.55 ± 0.77
GRF_MNB	0.464	0.424			15.81 ± 0.03	16.32 ± 0.61
GNB	0.483	0.43	0.00%	19.07%	13.34 ± 0.04	13.84 ± 0.56
GRF_GNB	0.485	0.524			14.58 ± 0.06	15.10 ± 0.49
SGD	0.165 ± 0.003	0.100 ± 0.003	14.30%	25.49%	9.92 ± 0.02	10.51 ± 0.43
GRF_SGD	0.331 ± 0.003	0.319 ± 0.002			11.00 ± 0.02	11.48 ± 0.30
Perceptron	0.132 ± 0.003	0.068 ± 0.001	34.83%	68.52%	8.02 ± 0.01	8.38 ± 0.19
GRF_Perceptron	0.121 ± 0.002	0.076 ± 0.003			10.03 ± 0.03	10.55 ± 0.22
PA	0.011	-0.076	100.00%	100.00%	7.87 ± 0.01	8.23 ± 0.17
GRF_PA	0.106	0.079			9.96 ± 0.02	10.32 ± 0.28
MLP	0.708 ± 0.023	0.734 ± 0.018	3.37%	13.82%	32.47 ± 0.14	33.06 ± 0.92
GRF_MLP	0.709 ± 0.023	0.777 ± 0.020			35.97 ± 0.22	39.78 ± 1.03

Table 4: Results for synthetic datasets *sineH_concept1* and *sineH_concept2*.

Stream Learners	Kappa		McNemar		Time	
	Weather	Electricity	Weather	Electricity	Weather	Electricity
KNN	0.354	0.610	6.10%	44.95%	4.52 ± 0.09	13.40 ± 0.68
GRF_KNN	0.364	0.640			6.15 ± 0.10	17.53 ± 0.90
HT	0.367	0.508	14.29%	40.00%	3.31 ± 0.05	8.29 ± 0.48
GRF_HT	0.402	0.491			7.82 ± 0.16	21.28 ± 1.27
HAT	0.300	0.736	20.70%	19.52%	4.82 ± 0.08	13.57 ± 3.42
GRF_HAT	0.339	0.758			10.65 ± 0.17	23.26 ± 1.80
MNB	0.000	0.002	0.00%	81.12%	5.76 ± 0.12	16.57 ± 1.18
GRF_MNB	0.000	0.347			7.01 ± 0.10	18.86 ± 1.32
GNB	0.307	0.378	25.98%	65.05%	4.85 ± 0.08	13.57 ± 0.97
GRF_GNB	0.366	0.271			6.63 ± 0.09	18.09 ± 1.22
SGD	0.403 ± 0.004	0.697 ± 0.002	6.18%	87.90%	3.71 ± 0.07	10.94 ± 0.73
GRF_SGD	0.392 ± 0.004	0.796 ± 0.002			5.31 ± 0.07	14.51 ± 0.91
Perceptron	0.382 ± 0.003	0.742 ± 0.002	4.68%	61.14%	2.98 ± 0.04	8.51 ± 0.59
GRF_Perceptron	0.380 ± 0.005	0.801 ± 0.002			4.91 ± 0.07	13.09 ± 0.90
PA	0.402 ± 0.001	0.752 ± 0.001	17.79%	72.00%	3.02 ± 0.04	8.26 ± 0.59
GRF_PA	0.378 ± 0.001	0.814 ± 0.001			4.99 ± 0.07	13.05 ± 0.89
MLP	0.405 ± 0.004	0.545 ± 0.005	39.70%	57.97%	12.12 ± 0.22	35.01 ± 2.42
GRF_MLP	0.442 ± 0.002	0.606 ± 0.002			14.75 ± 0.24	48.77 ± 3.27

Table 5: Results for the real datasets: *Weather* and *Electricity market*.

Stream Learners	Kappa		McNemar		Time	
	Moving sq.	SEA	Moving sq.	SEA	Moving sq.	SEA
KNN	0.993	0.623	99.96%	8.30%	17.54 ± 0.92	13.46 ± 0.17
GRF_KNN	0.931	0.615			21.82 ± 1.17	15.05 ± 0.19
HT	0.104	0.747	71.83%	28.00%	9.10 ± 0.53	5.73 ± 0.06
GRF_HT	0.160	0.726			37.95 ± 2.07	9.43 ± 0.12
HAT	0.656	0.640	80.02%	46.18%	14.69 ± 0.86	10.26 ± 0.13
GRF_HAT	0.407	0.695			58.19 ± 3.35	16.96 ± 0.22
MNB	0.060	0.284	57.78%	90.38%	19.87 ± 1.25	14.16 ± 0.20
GRF_MNB	0.090	0.419			21.06 ± 1.34	14.22 ± 0.22
GNB	0.102	0.755	82.80%	6.96%	18.63 ± 1.23	11.90 ± 0.18
GRF_GNB	0.102	0.754			22.52 ± 1.45	13.73 ± 0.22
SGD	0.408 ± 0.003	0.566 ± 0.004	100.00%	23.32%	31.68 ± 1.98	9.22 ± 0.13
GRF_SGD	0.770 ± 0.001	0.591 ± 0.002			32.28 ± 2.09	10.44 ± 0.15
Perceptron	0.338 ± 0.002	0.482 ± 0.002	100.00%	14.92%	25.33 ± 1.73	7.21 ± 0.11
GRF_Perceptron	0.795 ± 0.001	0.473 ± 0.004			29.43 ± 1.94	9.40 ± 0.14
PA	0.378 ± 0.003	0.395 ± 0.001	100.00%	55.24%	25.05 ± 1.68	7.03 ± 0.10
GRF_PA	0.815	0.466			29.61 ± 1.95	9.26 ± 0.13
MLP	0.966 ± 0.006	0.714 ± 0.001	22.36%	48.83%	42.71 ± 2.61	28.4 ± 0.44
GRF_MLP	0.981 ± 0.002	0.742 ± 0.002			47.60 ± 2.77	31.93 ± 0.55

Table 6: Results for the real datasets: *Moving squares* and *SEA*.

Stream Learners	Kappa	McNemar	Time
	Airlines		
KNN	0.092	100.00%	15.90 ± 0.62
GRF_KNN	1.000		24.28 ± 0.69
HT	1.000	1.34%	11.43 ± 0.14
GRF_HT	1.000		19.13 ± 0.25
HAT	1.000	1.34%	20.73 ± 0.31
GRF_HAT	1.000		28.07 ± 0.42
MNB	0.095	100.00%	16.88 ± 0.25
GRF_MNB	1.000		22.19 ± 0.34
GNB	1.000	0.00%	14.50 ± 0.23
GRF_GNB	1.000		21.88 ± 0.40
SGD	0.115 ± 0.002	100.00 %	10.82 ± 0.16
GRF_SGD	0.999		17.09 ± 0.24
Perceptron	0.116 ± 0.002	100.00%	8.25 ± 0.12
GRF_Perceptron	1.000		15.76 ± 0.23
PA	0.130	100.00 %	8.17 ± 0.12
GRF_PA	1.000		15.68 ± 0.20
MLP	0.711 ± 0.025	93.91%	34.55 ± 0.43
GRF_MLP	1.000		48.81 ± 1.16

Table 7: Results for the real dataset: *Airlines*.

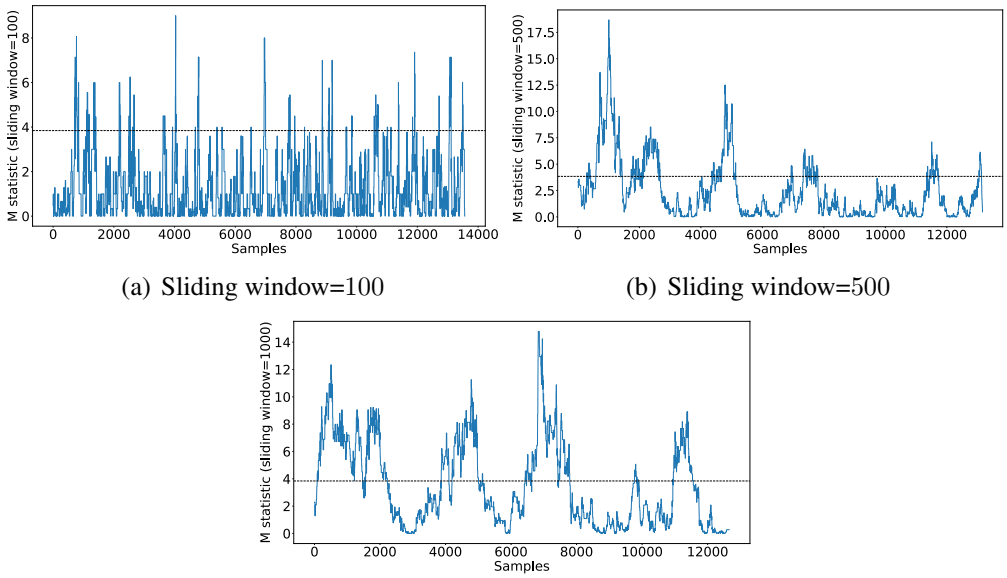
Stream Learners	Kappa	McNemar	Time
KNN	0.42	52.42%	12.73
GRF_KNN	0.49		14.87
HT	0.60	26.92%	6.76
GRF_HT	0.63		14.03
HAT	0.57	51.84%	12.65
GRF_HAT	0.59		22.18
MNB	0.13	54.78%	15.80
GRF_MNB	0.46		16.73
GNB	0.53	31.10%	13.44
GRF_GNB	0.53		16.03
SGD	0.33	51.79%	12.00
GRF_SGD	0.56		13.79
Perceptron	0.23	60.73%	9.46
GRF_Perceptron	0.48		12.63
PA	0.22	68.15%	9.36
GRF_PA	0.47		12.56
MLP	0.64	46.66%	32.39
GRF_MLP	0.69		35.88

Table 8: Summarized results for synthetic and real datasets. It contains the mean of Kappa statistics, the mean of the percentages of the McNemar tests in which the null hypothesis is rejected, and the mean of the processing times (seconds) over all datasets. The results in italics mean that there is the enough significance (more than 50% of the length of the dataset) to confirm that the application of GRFs population encoding shows a positive impact on the predictive performance of the SL methods.

7. Discussion

Regarding the statistical significance of the experiments, in Table 8 we can see where the null hypothesis is rejected more than 50% of the length of the datasets, which means that we may reject the null hypothesis in favor of the hypothesis that

the two stream learners have different performance when GRFs population encoding is applied. Concretely, in the case of KNN, HAT, MNB, SGD, Perceptron, and PA, we can confirm that the application of GRFs population encoding has shown a positive impact on the predictive performance of these SL methods. This predictive performance improvement is especially remarkable for Perceptron and PA, where the null hypothesis is rejected in more than 60% of the length of the datasets. For those cases in which the null hypothesis is rejected less than 50%, we can also underline that the predictive performance of the SL methods has not been decreased, therefore the application of GRFs population encoding has not impacted negatively on them. For the McNemar test we have chosen a sliding window of 500 samples. Nevertheless, the experiments in (Gama, Sebastião & Rodrigues, 2013) pointed out that for different sizes of sliding windows we could obtain different results about the significance of the differences, as Figure 8 reflects.



(a) Sliding window=100

(b) Sliding window=500

(c) Sliding window=1000

Figure 8: The evolution of the McNemar statistic between the MLP stream learner and its version with GRFs population encoding scheme. The dotted line is the threshold for a significance level of 95%. The null hypothesis is rejected 7.47% (a), 28.81% (b), and 50.23% (c).

Once the statistical significance has been confirmed, we now turn the focus on the predictive performance improvement. In all cases where the statistical significance is evident, the difference between the two stream learners is notable:

0.33 in MNB, 0.23 in SGD, 0.25 in Perceptron, 0.25 in PA. For KNN and HAT the differences are less considerable (0.07 and 0.02 respectively), but also solid due to the number of repetitions (25) of each experiment.

In what refers to the time processing, in all cases the application of GRFs population encoding increases the required time to process the stream data, but with more or less relevancy depending on the stream learner. In the case of MNB we increase the predictive performance in 0.33 by adding only a 5.88% extra for the processing time, thus we always recommend the application of the DP technique. For PA and SGD the increment in the predictive performance is 0.25 and 0.23 respectively, while the extra required processing time is 10.77% and 14.91% respectively; in these cases, we also recommend the application of the DP technique. For Perceptron the increment of the predictive performance is also notable (0.25), but we should consider the application of the DP technique carefully because we have to assume a 33.50% extra for the time processing. Finally, for KNN (an increase of 0.07 in the predictive performance with a 16.81% increment in the processing time) and HAT (an increase of 0.02 in the predictive performance with a 75.33% increment in the processing time) the application of the DP technique is less recommendable, but we may find some applications fields in which prevails the predictive performance over the processing time.

This general increment in the processing time of the SL methods makes sense when we consider that GRFs population encoding scheme acts in this study as a DP technique, adding extra calculous to the SL method (more GRFs implies more cut points per feature in the new representation, and then a new larger real-valued vector. See Algorithm 1). Therefore, we will have to tune this parameter good enough to achieve an increment in the predictive performance without drastically penalizing the processing time of the SL algorithm.

8. Conclusions and Future Work

This study has elaborated on a new approach for applying a GRFs population encoding scheme to the input data of any stream learning method, augmenting the representativeness of the incoming stimuli, and thus achieving a significant improvement in the predictive performance of the stream learners. A wide variety of synthetic and real datasets have been used for testing a large number of well-known stream learning methods, constituting a complete set of experiments. In order to be sure about the statistical significance of these experiments, McNemar tests have been used to assess the differences in predictive performance of the stream learning methods. This scheme can be seen as a pre-processing technique to be carried out every time a new sample arrives to the stream learning process. Although this scheme increases the processing time at different levels depending

on the stream learner method, it has been shown how their predictive performance can boosted, making it worthwhile to apply in those cases where the processing time between samples allows for it. In those cases where the scheme does not reflect a statistical significance, the study has confirmed that the application of this scheme does not harm the predictive performance of the stream learner, only the processing time at different levels.

Future efforts should be invested on studying how the application of this GRFs population encoding scheme can affect to the drift adaptation, because it could help the stream learner to adapt better to the drift and to recover its predictive performance faster after drift is detected. As it has been mentioned in Section 3, the choice of the GRFs parameters may have a different impact on the features representation of each class; the possibility of applying different values for the GRFs parameters to each class should be further analyzed. Finally, this preprocessing technique could be also applied to batch learning problems, and we encourage other researcher to apply this technique to batch learning methods with batch datasets.

Acknowledgements

This work was supported by the EU project *iDev40*. This project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 783163. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Germany, Belgium, Italy, Spain, Romania. It has also been supported by the Basque Government (Spain) through the project *VIRTUAL* (KK-2018/00096).

Appendix A. Parameter Configuration for the SL Methods

Note: In order to have more detail about the meaning of the parameters and their default values, please we recommend to check the frameworks mentioned in subsection 5.1.

Stream Learners	Electricity market	Moving Squares
KNN	n_neighbors=3, max_window_size=10, leaf_size=2, pre-training_size=11,000, M_sw_size=500	n_neighbors=20, max_window_size=50, leaf_size=2, pre-training_size=12,500, M_sw_size=500
GRF_KNN	KNN + GRF params (gamma=2.0, n_GRFs=5)	KNN + GRF params (gamma=2.0, n_GRFs=11)
HT	parameters by default	parameters by default
	pre-training_size=11,000, M_sw_size=500	pre-training_size=12,500, M_sw_size=500
GRF_HT	HT + GRF params (gamma=2.0, n_GRFs=5)	HT + GRF params (gamma=2.0, n_GRFs=11)
HAT	HT + ADWIN params (delta=0.002, f=32)	HT + ADWIN params (delta=0.002, f=32)
	pre-training_size=11,000, M_sw_size=500	pre-training_size=12,500, M_sw_size=500
GRF_HAT	HAT + GRF params (gamma=2.0, n_GRFs=5)	HAT + GRF params (gamma=2.0, n_GRFs=11)
MNB	alpha=1.0, fit_prior=True	alpha=1.0, fit_prior=True
	pre-training_size=11,000, M_sw_size=500	pre-training_size=12,500, M_sw_size=500
GRF_MNB	MNB + GRF params (gamma=2.0, n_GRFs=5)	MNB + GRF params (gamma=2.0, n_GRFs=11)
GNB	var_smoothing=$1e-9$	var_smoothing= $1e-9$
	pre-training_size=11,000, M_sw_size=500	pre-training_size=12,500, M_sw_size=500
GRF_GNB	GNB + GRF params (gamma=2.0, n_GRFs=5)	GNB + GRF params (gamma=2.0, n_GRFs=11)
SGD	parameters by default with n_iter=1	parameters by default with n_iter=1
	pre-training_size=11,000, M_sw_size=500	pre-training_size=12,500, M_sw_size=500
GRF_SGD	SGD + GRF params (gamma=2.0, n_GRFs=5)	SGD + GRF params (gamma=2.0, n_GRFs=11)
Perceptron	parameters by default with loss='perceptron' and n_iter=1	parameters by default with loss='perceptron' and n_iter=1
	pre-training_size=11,000, M_sw_size=500	pre-training_size=12,500, M_sw_size=500
GRF_Perceptron	Perceptron + GRF params (gamma=2.0, n_GRFs=5)	Perceptron + GRF params (gamma=2.0, n_GRFs=11)
PA	parameters by default with n_iter=1	parameters by default with n_iter=1
	pre-training_size=11,000, M_sw_size=500	pre-training_size=12,500, M_sw_size=500
GRF_PA	PA + GRF params (gamma=2.0, n_GRFs=5)	PA + GRF params (gamma=2.0, n_GRFs=11)
MLP	parameters by default with max_iter=1	parameters by default with max_iter=1
	pre-training_size=11,000, M_sw_size=500	pre-training_size=12,500, M_sw_size=500
GRF_MLP	MLP + GRF params (gamma=2.0, n_GRFs=5)	MLP + GRF params (gamma=2.0, n_GRFs=11)

Table A.9: Parameters configuration of the SL methods for datasets *Electricity market* and *Moving squares*.

Stream Learners	Weather	SEA
KNN	n_neighbors=5, max_window_size=50, leaf_size=2, pre-training_size=4, 500, M_sw_size=500	n_neighbors=15, max_window_size=100, leaf_size=2, pre-training size=10, 000, M_SW_size=500
GRF_KNN	KNN + GRF params (gamma=2.0, n_GRFs=3)	KNN + GRF params (gamma=2.0, n_GRFs=3)
HT	parameters by default	parameters by default
	pre-training_size=4, 500, M_sw_size=500	pre-training_size=10, 000, M_sw_size=500
GRF_HT	HT + GRF params (gamma=2.0, n_GRFs=3)	HT + GRF params (gamma=2.0, n_GRFs=3)
HAT	HT + ADWIN params (delta=0.002, f=32)	HT + ADWIN params (delta=0.002, f=32)
	pre-training_size=4, 500, M_sw_size=500	pre-training_size=10, 000, M_sw_size=500
GRF_HAT	HAT + GRF params (gamma=2.0, n_GRFs=3)	HAT + GRF params (gamma=2.0, n_GRFs=3)
MNB	alpha=1.0, fit_prior=True	alpha=1.0, fit_prior=True
	pre-training_size=4, 500, M_sw_size=500	pre-training_size=10, 000, M_sw_size=500
GRF_MNB	MNB + GRF params (gamma=2.0, n_GRFs=3)	MNB + GRF params (gamma=2.0, n_GRFs=3)
GNB	var_smoothing=$1e-9$	var_smoothing=$1e-9$
	pre-training_size=4, 500, M_sw_size=500	pre-training_size=10, 000, M_sw_size=500
GRF_GNB	GNB + GRF params (gamma=2.0, n_GRFs=3)	GNB + GRF params (gamma=2.0, n_GRFs=3)
SGD	parameters by default with n_iter=1	parameters by default with n_iter=1
	pre-training_size=4, 500, M_sw_size=500	pre-training_size=10, 000, M_sw_size=500
GRF_SGD	SGD + GRF params (gamma=2.0, n_GRFs=3)	SGD + GRF params (gamma=2.0, n_GRFs=3)
Perceptron	parameters by default with loss='perceptron' and n_iter=1	parameters by default with loss='perceptron' and n_iter=1
	pre-training_size=4, 500, M_sw_size=500	pre-training_size=10, 000, M_sw_size=500
GRF_Perceptron	Perceptron + GRF params (gamma=2.0, n_GRFs=3)	Perceptron + GRF params (gamma=2.0, n_GRFs=3)
PA	parameters by default with n_iter=1	parameters by default with n_iter=1
	pre-training_size=4, 500, M_sw_size=500	pre-training_size=12, 500, M_sw_size=500
GRF_PA	PA + GRF params (gamma=2.0, n_GRFs=3)	PA + GRF params (gamma=2.0, n_GRFs=3)
MLP	parameters by default with max_iter=1	parameters by default with max_iter=1
	pre-training_size=4, 500, M_sw_size=500	pre-training_size=12, 500, M_sw_size=500
GRF_MLP	MLP + GRF params (gamma=2.0, n_GRFs=3)	MLP + GRF params (gamma=2.0, n_GRFs=3)

Table A.10: Parameters configuration of the SL methods for datasets *Weather* and *SEA*.

Stream Learners	Airlines	Synthetic datasets
KNN	n_neighbors=7, max_window_size=70, leaf_size=2, pre-training_size=12, 500, M_sw_size=500	n_neighbors=3, max_window_size=10, leaf_size=2, pre-training_size=12, 500, M_sw_size=500
GRF_KNN	KNN + GRF params (gamma=2.0, n_GRFs=3)	KNN + GRF params (gamma=2.0, n_GRFs=3)
HT	parameters by default	parameters by default
	pre-training_size=12, 500, M_sw_size=500	pre-training_size=12, 500, M_sw_size=500
GRF_HT	HT + GRF params (gamma=2.0, n_GRFs=3)	HT + GRF params (gamma=2.0, n_GRFs=3)
HAT	HT + ADWIN params (delta=0.002, f=32)	HT + ADWIN params (delta=0.002, f=32)
	pre-training_size=12, 500, M_sw_size=500	pre-training_size=12, 500, M_sw_size=500
GRF_HAT	HAT + GRF params (gamma=2.0, n_GRFs=3)	HAT + GRF params (gamma=2.0, n_GRFs=3)
MNB	alpha=1.0, fit_prior=True	alpha=1.0, fit_prior=True
	pre-training_size=12, 500, M_sw_size=500	pre-training_size=12, 500, M_sw_size=500
GRF_MNB	MNB + GRF params (gamma=2.0, n_GRFs=3)	MNB + GRF params (gamma=2.0, n_GRFs=3)
GNB	var_smoothing=$1e-9$	var_smoothing=$1e-9$
	pre-training_size=12, 500, M_sw_size=500	pre-training_size=12, 500, M_sw_size=500
GRF_GNB	GNB + GRF params (gamma=2.0, n_GRFs=3)	GNB + GRF params (gamma=2.0, n_GRFs=3)
SGD	parameters by default with n_iter=1	parameters by default with n_iter=1
	pre-training_size=12, 500, M_sw_size=500	pre-training_size=12, 500, M_sw_size=500
GRF_SGD	SGD + GRF params (gamma=2.0, n_GRFs=3)	SGD + GRF params (gamma=2.0, n_GRFs=3)
Perceptron	parameters by default with loss='perceptron' and n_iter=1	parameters by default with loss='perceptron' and n_iter=1
	pre-training_size=12, 500, M_sw_size=500	pre-training_size=12, 500, M_sw_size=500
GRF_Perceptron	Perceptron + GRF params (gamma=2.0, n_GRFs=3)	Perceptron + GRF params (gamma=2.0, n_GRFs=3)
PA	parameters by default with n_iter=1	parameters by default with n_iter=1
	pre-training_size=12, 500, M_sw_size=500	pre-training_size=12, 500, M_sw_size=500
GRF_PA	PA + GRF params (gamma=2.0, n_GRFs=3)	PA + GRF params (gamma=2.0, n_GRFs=3)
MLP	parameters by default with max_iter=1	parameters by default with max_iter=1
	pre-training_size=12, 500, M_sw_size=500	pre-training_size=12, 500, M_sw_size=500
GRF_MLP	MLP + GRF params (gamma=2.0, n_GRFs=3)	MLP + GRF params (gamma=2.0, n_GRFs=3)

Table A.11: Parameters configuration of the SL methods for datasets *Airlines* and *Synthetic datasets* (*circle_concept1*, *circle_concept2*, *line_concept1*, *line_concept2*, *sine_concept1*, *sine_concept2*, *sineH_concept1*, and *sineH_concept2*).

Bibliography

Barros, R. S. M., & Santos, S. G. T. C. (2018). A large-scale comparison of concept drift detectors. *Information Sciences*, *451*, 348–370.

Bifet, A., de Francisci Morales, G., Read, J., Holmes, G., & Pfahringer, B. (2015). Efficient online evaluation of big data stream classifiers. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 59–68). ACM.

Bifet, A., & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining* (pp. 443–448). SIAM.

Bifet, A., & Gavaldà, R. (2009). Adaptive learning from evolving data streams. In *International Symposium on Intelligent Data Analysis* (pp. 249–260). Springer.

Bifet, A., Gavaldà, R., Holmes, G., & Pfahringer, B. (2018). *Machine Learning for Data Streams with Practical Examples in MOA*. MIT Press. `https://moa.cms.waikato.ac.nz/book/`.

Bifet, A., Holmes, G., Pfahringer, B., & Frank, E. (2010). Fast perceptron decision tree learning from evolving data streams. In *Pacific-Asia conference on knowledge discovery and data mining* (pp. 299–310). Springer.

Bohte, S. M., Kok, J. N., & La Poutre, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, *48*, 17–37.

Chan, T. F., Golub, G. H., & LeVeque, R. J. (1982). Updating formulae and a pairwise algorithm for computing sample variances. In *COMPSTAT 1982 5th Symposium held at Toulouse 1982* (pp. 30–41). Springer.

Chen, M., Mao, S., & Liu, Y. (2014). Big data: A survey. *Mobile networks and applications*, *19*, 171–209.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychological measurement*, *20*, 37–46.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., & Singer, Y. (2006). Online passive-aggressive algorithms. *Journal of Machine Learning Research*, *7*, 551–585.

Dasarathy, B. V. (1991). Nearest neighbor ({NN}) norms:{NN} pattern classification techniques, .

De Francisci Morales, G., Bifet, A., Khan, L., Gama, J., & Fan, W. (2016). Iot big data stream mining. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 2119–2120). ACM.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, *10*, 1895–1923.

Elwell, R., & Polikar, R. (2011). Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, *22*, 1517–1531.

Freund, Y., & Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine learning*, *37*, 277–296.

Gama, J., Sebastião, R., & Rodrigues, P. P. (2013). On evaluating stream learning algorithms. *Machine learning*, *90*, 317–346.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, *46*, 44.

García, S., Ramírez-Gallego, S., Luengo, J., Benítez, J. M., & Herrera, F. (2016). Big data preprocessing: methods and prospects. *Big Data Analytics*, *1*, 9.

Gerstner, W., & Kistler, W. M. (2002). *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press.

Gonçalves Jr, P. M., de Carvalho Santos, S. G., Barros, R. S., & Vieira, D. C. (2014). A comparative study on concept drift detectors. *Expert Systems with Applications*, *41*, 8144–8156.

Harries, M., & Wales, N. S. (1999). Splice-2 comparative evaluation: Electricity pricing, .

Hinton, G. E. (1990). Connectionist learning procedures. In *Machine Learning, Volume III* (pp. 555–610). Elsevier.

Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 97–106). ACM.

Ikonomovska, E., Gama, J., & Džeroski, S. (2011). Learning model trees from evolving data streams. *Data mining and knowledge discovery*, *23*, 128–168.

Kasabov, N. K. (2007). *Evolving connectionist systems: the knowledge engineering approach*. Springer Science & Business Media.

Kasabov, N. K. (2018). *Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence*. Springer.

Lobo, J. L., Del Ser, J., Laña, I., Bilbao, M. N., & Kasabov, N. (2018a). Drift detection over non-stationary data streams using evolving spiking neural networks. In *International Symposium on Intelligent and Distributed Computing* (pp. 82–94). Springer.

Lobo, J. L., Laña, I., Del Ser, J., Bilbao, M. N., & Kasabov, N. (2018b). Evolving spiking neural networks for online learning over drifting data streams. *Neural Networks*, .

Losing, V., Hammer, B., & Wersing, H. (2016). Knn classifier with self adjusting memory for heterogeneous concept drift. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on* (pp. 291–300). IEEE.

McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, *12*, 153–157.

Minku, L. L., White, A. P., & Yao, X. (2010). The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on knowledge and Data Engineering*, *22*, 730–742.

Minku, L. L., & Yao, X. (2012). Ddd: A new ensemble approach for dealing with concept drift. *IEEE transactions on knowledge and data engineering*, *24*, 619–633.

Montiel, J., Read, J., Bifet, A., & Abdessalem, T. (2018). Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, *19*, 1–5. URL: `http://jmlr.org/papers/v19/18-251.html`.

Polikar, R., Upda, L., Upda, S. S., & Honavar, V. (2001). Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, *31*, 497–508.

Ponulak, F. (2005). Resume-new supervised learning method for spiking neural networks. *Institute of Control and Information Engineering, Poznan University of Technology*, *42*.

Ramírez-Gallego, S., Krawczyk, B., García, S., Woźniak, M., & Herrera, F. (2017). A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing*, *239*, 39–57.

Robbins, H., & Monro, S. (1985). A stochastic approximation method. In *Herbert Robbins Selected Papers* (pp. 102–109). Springer.

Schliebs, S., & Kasabov, N. (2013). Evolving spiking neural networka survey. *Evolving Systems*, *4*, 87–98.

Street, W. N., & Kim, Y. (2001). A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 377–382). ACM.

Thorpe, S., & Gautrais, J. (1998). Rank order coding. In *Computational neuroscience* (pp. 113–118). Springer.

Wang, J., Belatreche, A., Maguire, L. P., & McGinnity, T. M. (2017). Spiketemp: An enhanced rank-order-based learning approach for spiking neural networks with adaptive structure. *IEEE transactions on neural networks and learning systems*, *28*, 30–43.

Webb, G. I., Hyde, R., Cao, H., Nguyen, H. L., & Petitjean, F. (2016). Characterizing concept drift. *Data Mining and Knowledge Discovery*, *30*, 964–994.

Zhang, H. (2004). The optimality of naive bayes. *AA*, *1*, 3.