Research Article

Data Stream Classification Based on the Gamma Classifier

Abril Valeria Uriarte-Arcia,¹ Itzamá López-Yáñez,² Cornelio Yáñez-Márquez,¹ João Gama,³ and Oscar Camacho-Nieto²

¹Neural Networks and Unconventional Computing Lab/Alpha-Beta Group, Centro de Investigación en Computación, Instituto Politécnico Nacional, Avenida Juan de Dios Bátiz, Colonia Nuevo Industrial Vallejo, Delegación Gustavo A. Madero, 07738 Mexico City, DF, Mexico

²Intelligent Computing Lab/Alpha-Beta Group, Centro de Innovación y Desarrollo Tecnológico en Cómputo, Instituto Politécnico Nacional, Avenida Juan de Dios Bátiz, Colonia Nuevo Industrial Vallejo, Delegación Gustavo A. Madero, 07700 Mexico City, DF, Mexico

³LIAAD-INESC INESC TEC and Faculty of Economics, University of Porto, Rua Dr. Roberto Frias 378, 4200-378 Porto, Portugal

Correspondence should be addressed to Abril Valeria Uriarte-Arcia; auriarteb10@sagitario.cic.ipn.mx

Received 13 April 2015; Revised 7 July 2015; Accepted 29 July 2015

Academic Editor: Konstantinos Karamanos

Copyright © 2015 Abril Valeria Uriarte-Arcia et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The ever increasing data generation confronts us with the problem of handling online massive amounts of information. One of the biggest challenges is how to extract valuable information from these massive continuous data streams during single scanning. In a data stream context, data arrive continuously at high speed; therefore the algorithms developed to address this context must be efficient regarding memory and time management and capable of detecting changes over time in the underlying distribution that generated the data. This work describes a novel method for the task of pattern classification over a continuous data stream based on an associative model. The proposed method is based on the Gamma classifier, which is inspired by the Alpha-Beta associative memories, which are both supervised pattern recognition models. The proposed method is capable of handling the space and time constrain inherent to data stream scenarios. The Data Streaming Gamma classifier (DS-Gamma classifier) implements a sliding window approach to provide concept drift detection and a forgetting mechanism. In order to test the classifier, several experiments were performed using different data stream scenarios with real and synthetic data streams. The experimental results show that the method exhibits competitive performance when compared to other state-of-the-art algorithms.

1. Introduction

In recent years, technological advances have promoted the generation of a vast amount of information from different areas of knowledge: sensor networks, financial data, fraud detection, and web data, among others. According to the study performed by IDC (International Data Corporation) [1], the digital universe in 2013 was estimated in 4.4 trillion gigabytes. From this digital data, only 22% would be a candidate for analysis, while the available storage capacity could hold just 33% of the generated information. Under this scenario, the extraction of knowledge is becoming a very challenging task. We are now confronted with the problem of handling very large datasets and even with the possibility of

an infinite data flow. Therefore the algorithms developed to address this context, unlike traditional ones, must meet some constraints, as defined in [2], work with a limited amount of time, and use a limited amount of memory, and one or only few pass over the data. They also should be capable of reacting to concept drift, that is, changes in the distribution of the data over time. In [3], more details of the requirements to be considered for data streams algorithms can be found.

As the idea suggests, a data stream can roughly be thought of as an ordered endless sequence of data items, where the input arrives more or less continuously as time progresses [4]. The extensive research on data stream algorithms performed over the last years has produced a large variety of techniques for data stream problems. Several relevant data mining tasks

have been addressed: clustering, classification, prediction, and concept drift detection. Data stream algorithms can be roughly divided into single classifier approach and ensemble approach. The single model approach can be further divided into model based approach and instance based approach. Model based approaches have the disadvantage of the computational cost of updating the model every time that a new element of the stream arrives. In instance based approaches just the representative subset of elements (case base) from the data stream have to be adapted; this does not come for free since the computational cost is reflected during the classification stage [4]. The single classifier approach includes several data mining techniques such as decision trees, decision rules, instance based learners (IBL), and neural networks, among others. A more detailed taxonomy of data stream algorithms can be found in [5].

Decision trees have been widely used in data stream classification [6–9]. One of the most representative models for data stream classification using decision trees was proposed by Domingos and Hulten [6, 7] and has been the inspiration for several of the tree models used for data stream mining. In [6], the authors proposed a Very Fast Decision Tree (VFDT) learner that uses Hoeffding bounds, which guarantee that the tree can be trained in constant time. An extension of this work was presented in [7] where the decision trees are built from a sliding window of fixed size. Trees are also commonly used as base learners for ensemble approaches.

Decision rules models have also been applied to data stream context. One of the first methods proposed for rule induction in an online environment was FLORA (FLOating Rough Approximation) [10]. To deal with concept drift and hidden contexts, this framework uses three techniques: keep only a window of trusted examples, store concept descriptions to reuse them when a previous context reappears, and use heuristics to control these two functions. Another relevant work was presented by Ferrer-Troyano et al. in [11]. The method is called FACIL (Fast and Adaptive Classifier by Incremental Learning), an incremental classifier based on decision rules. The classifier uses a partial instance memory where examples are rejected if they do not describe a decision boundary.

Algorithms for data stream context have to be incremental and highly adaptive. In IBL algorithms incremental learning and model adaptation are very simple since these methods come down to adapt the case base (i.e., representative examples). This allows obtaining a flexible and easily updated model, without complex computations that optimize the update and classification times [12]. Beringer and Hüllermeier proposed an IBL algorithm for data stream classification [4]; the system maintains an implicit concept description in the form of a case base that is updated taking into account three indicators: temporal relevance, spatial relevance, and consistency. On the arrival of a new example, this example is incorporated into the case base and then a replacement policy is applied based on the three previously mentioned indicators. In [12], a new technique named Similarity-Based Data Stream Classifier (SimC) is introduced. The system maintains a representative and small

set of information in order to conserve the distribution of classes over the stream. It controls noise and outliers applying an insertion/removal policy designed for retaining the best representation of the knowledge using appropriate estimators. Shaker and Hüllermeier presented an instance based algorithm that can be applied to classification and regression problems [13]. The algorithm is called IBLStream and it optimizes the composition and size of the case base autonomously. On arrival of a new example, this example is first added to the case base and then it is checked whether other examples might be removed either since they have become redundant or since they are noisy data.

For data stream scenarios, neural network-based models have hardly been used; one of the main issues of this approach is the weights update caused by the arrival of a new example. The influence on the network of this single example cannot simply be canceled later on; at best, it can be reduced gradually over the course of time [12]. However there are some works where neural networks are used for data stream tasks [14], where an online perceptron is used to classify non-stationary imbalance data streams. They have also been used as base learners for ensemble methods [15, 16].

The ensemble methods have also been extensively used in data stream classification. The ensemble approach, as its name implies, maintains in memory an ensemble of multiple models whose individual predictions are combined—generally by average or voting techniques—to output a final prediction [2]. The main idea behind an ensemble classifier is based on the concept that different learning algorithms explore different search spaces and evaluation of the hypothesis [3]. Different classifiers can be used as base learners in the ensemble methods, for instance, tree [17–19], Naïve Bayes [20, 21], neural networks [15], and KNN [22].

SVMs have been used for different tasks in the data streams scenarios such as sentiment analysis, fault detection and prediction, medical applications, spam detection, and multilabel classification. In [23], a fault-detection system based on data stream prediction is proposed. SVM algorithm is used to classify the incoming data and trigger a fault alarm in case the data is considered abnormal. Sentiment analysis has also been addressed using SVM methods. Kranjc et al. [24] introduce a cloud-based scientific workflow platform, which is able to perform online dynamic adaptive sentiment analysis on twitter data. They used a linear SVM for sentiment classification. While this work does not focus on a specific topic, Smailović et al. [25] present a methodology that analyzes whether the sentiment expressed in twitter feeds, which discuss selected companies, can indicate their stock price changes. They propose the use of the Pegasos SVM [26] to classify the tweets as positive, negative, and neutral. Then the tweets classified as positive were used to calculate a sentiment probability for each day and this is used in a Granger causality analysis to correlate positive sentiment probability and stock closing price. In [27], a stream clustering SVM algorithm to treat spam identification was presented. While in general this problem has previously been addressed as a classification problem, they handle it as an anomaly detection problem. Krawczyk and Woźniak [28] proposed a version of

incremental One-Class Support Vector Machine that assigns weights to each object according to its level of significance. They also introduce two schemes for estimating weights for new, incoming data and examine their usefulness.

Another well studied issue in the data stream context is concept drift detection. The change in the concept might occur due to changes in hidden variables affecting the phenomena or changes in the characteristics of the observed variables. According to [29], two main tactics are used in concept drift detection: (1) monitoring the evolution of performance indicators using statistical techniques and (2) monitoring distribution of two different time windows. Some algorithms use methods that adapt the decision model at regular intervals without taking into account if change really happened; this involves extra processing time. On the other hand, explicit change methods can detect the point of change, quantify the change, and take the needed actions. In general, the methods for concept drift detection can be integrated with different base learners. Some methods related to this issue can be found in [30–32].

Some other issues related to data stream classification have also been addressed, such as unbalanced data stream classification [14, 15, 19], classification in the presence of uncertain data [8], classification with unlabeled instances [8, 33, 34], novel class detection [22, 35], and multilabel classification [36].

In this paper we describe a novel method for the task of pattern classification over a continuous data stream based on an associative model. The proposed method is based on the Gamma classifier [37], which is a supervised pattern recognition model inspired on the Alpha-Beta associative memories [38]. The proposal has been developed to work in data stream scenarios using a sliding window approach and it is capable of handling the space and time constraints inherent to this type of scenarios. Various methods have been designed for data stream classification, but to our knowledge the associative approach has never been used for this kind of problems. The main contribution of this work is the implementation of a novel method for data stream classification based on an associative model.

The rest of this paper is organized as follows. Section 2 describes all the materials and methods needed to develop our proposal. Section 3 describes how the experimental phase was conducted and discusses the results. Some conclusions are presented in Section 4, and finally the Acknowledgments and References are included.

2. Materials and Methods

2.1. Gamma Classifier. This classifier is a supervised method whose name is derived from the similarity operator that it uses: the generalized Gamma operator. This operator takes as input two binary vectors, x and y , and a positive integer θ and returns 1 if both vectors are similar (θ being the degree of allowed dissimilarity) or 0 otherwise. The Gamma operator uses other operators (namely, α , β , and u_β) that will be introduced first. The rest of this section is strongly based on [37, 38].

TABLE 1: The Alpha (α) and Beta (β) operators.

(a)		
$\alpha: A \times A \rightarrow B$		
x	y	$\alpha(x, y)$
0	0	1
0	1	0
1	0	2
1	1	1
(b)		
$\beta: B \times A \rightarrow A$		
x	y	$\beta(x, y)$
0	0	0
0	1	0
1	0	0
1	1	1
2	0	1
2	1	1

Definition 1 (Alpha and Beta operators). Given the sets $A = \{0, 1\}$ and $B = \{0, 1, 2\}$, the Alpha (α) and Beta (β) operators are defined in a tabular form as shown in Table 1.

Definition 2 (Alpha operator applied to vectors). Let $x, y \in A^n$ with $n \in \mathbb{Z}^+$ be the input column vectors. The output for $\alpha(x, y)$ is n -dimensional vector, whose i th component ($i = 1, 2, \dots, n$) is computed as follows:

$$\alpha(x, y)_i = \alpha(x_i, y_i). \quad (1)$$

Definition 3 (u_β operator). Considering the binary pattern $x \in A^n$ as input, this unary operator gives the following non-negative integer as output and is calculated as follows:

$$u_\beta(x) = \sum_{i=1}^n \beta(x_i, x_i). \quad (2)$$

Thus, if $x = [1 \ 1 \ 0 \ 0 \ 1 \ 0]$ then

$$\begin{aligned} u_\beta(x) &= \beta(1, 1) + \beta(1, 1) + \beta(0, 0) + \beta(0, 0) \\ &\quad + \beta(1, 1) + \beta(0, 0) = 1 + 1 + 0 + 0 + 1 + 0 \\ &= 3. \end{aligned} \quad (3)$$

Definition 4 (pruning operator). Let $x \in A^n$ and $y \in A^m$; let $n, m \in \mathbb{Z}^+$, $n < m$, be two binary vectors; then y pruned by x , denoted by $y\|_x$, is n -dimensional binary vector whose i th component is defined as follows:

$$(y\|_x)_i = y_{i+m-n}, \quad (4)$$

where $(i = 1, 2, \dots, n)$.

For instance, let $y = [1 \ 1 \ 1 \ 1 \ 0 \ 1]$, $x = [0 \ 1 \ 0]$, $m = 6$, and then

$$\begin{aligned} (y\|_x)_1 &= y_{1+6-3} = y_4, \\ (y\|_x)_2 &= y_{2+6-3} = y_5, \\ (y\|_x)_3 &= y_{3+6-3} = y_6, \\ (y\|_x) &= [1 \ 0 \ 1]. \end{aligned} \quad (5)$$

Thus, only the 4th, 5th, and 6th elements of y will be used in $(y\|_x)_i$. Notice that the first $m - n$ components of y are discarded when building $y\|_x$.

The Gamma operator needs a binary vector as input. In order to deal with real and/or integer vectors a method to represent this vector in a binary form is needed. In this work, we will use the modified Johnson-Möbius code [39], which is a variation of the classical Johnson-Möbius code. To convert a set of real/integer numbers into a binary representation we will follow these steps:

- (1) Let R be a set of real numbers such that $R = \{r_1, r_2, \dots, r_i, \dots, r_n\}$, where $n \in \mathbb{R}$.
- (2) If there is one or more negative numbers in the set, create a new set from subtracting the minimum (i.e., r_i) from each number in R , obtaining a new set T with only nonnegative real numbers; $T = \{t_1, t_2, \dots, t_i, \dots, t_n\}$, where $t_j = r_j - r_i \ \forall j \in \{1, 2, \dots, n\}$ and particularly $t_i = 0$.
- (3) Choose a fixed number d and truncate each number of the set T to have only d decimal positions.
- (4) Scale up all the numbers of the set obtained in the previous step, by multiplying all numbers by 10^d , in order to leave only nonnegative integer numbers. $E = \{e_1, e_2, \dots, e_m, \dots, e_n\}$, where $e_j = t_j \cdot 10^d \ \forall j \in \{1, 2, \dots, n\}$ and e_m is the maximum value of the set E .
- (5) For each element $e_j \in E$ with $j = 1, 2, \dots, n$, concatenate $(e_m - e_j)$ zeroes with e_j ones.

For instance, let $R \subset \mathbb{R}$ be defined as $R = \{2.4, -0.7, 1.8, 1.5\}$; now let us use the modified Johnson-Möbius code to convert the elements of R into binary vectors.

- (1) Subtract the minimum. Since -0.7 is the minimum in R , the members of the transformed set T are obtained by subtracting -0.7 from each member in R . $t_i = d_i - (-0.7) = d_i + 0.7$. Consider

$$\begin{aligned} D &\longrightarrow T \subset \mathbb{R}, \\ T &= \{3.1, 0, 2.5, 2.2\}. \end{aligned} \quad (6)$$

- (2) Scale up the numbers. We select $d = 1$, so each number is truncate to 1 decimal, and then multiply by 10 ($10^d = 10^1 = 10$). One has

$$\begin{aligned} T &\longrightarrow E \subset \mathbb{Z}^+, \\ E &= \{31, 0, 25, 22\}. \end{aligned} \quad (7)$$

- (3) Concatenate $(e_m - e_j)$ zeroes with e_j ones, where e_m is the maximum number in E and e_j is the current number to be coded. Given that the maximum number in E is 31, all the binary vectors have 31 components; that is, they all are 31 bits long. For example, $e_4 = 22$ is converted into its binary representation c_4 by appending $e_m - e_4 = 31 - 22 = 9$ zeroes, followed by $e_4 = 22$ ones, which gives the final vector "0000000001111111111111111111111". Consider

$$\begin{aligned} E &\longrightarrow C = \{c_i \mid c_i \in A^{31}\}, \\ C &= \left\{ \begin{bmatrix} [1111111111111111111111111111111] \\ [0000000000000000000000000000000] \\ [000000111111111111111111111111111] \\ [000000000111111111111111111111111] \end{bmatrix} \right\}. \end{aligned} \quad (8)$$

Definition 5 (Gamma operator). The similarity Gamma operator takes two binary patterns, $x \in A^n$ and $y \in A^m$, $n, m \in \mathbb{Z}^+$, $n \leq m$, and a nonnegative integer θ as input and outputs a binary number, for which there are two cases.

Case 1. If $n = m$ then the output is computed according to the following:

$$\gamma_g(x, y, \theta) = \begin{cases} 1, & \text{if } m - u_\beta[\alpha(x, y) \bmod 2] \leq \theta, \\ 0, & \text{otherwise,} \end{cases} \quad (9)$$

where mod denotes the usual modulo operation.

Case 2. If $n < m$, then the output is computed using $y\|_x$ instead of y as follows:

$$\begin{aligned} \gamma_g(x, y, \theta) &= \begin{cases} 1, & \text{if } m - u_\beta[\alpha(x, y\|_x) \bmod 2] \leq \theta, \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \quad (10)$$

In order to better illustrate how the generalized Gamma operator works, let us work through some examples of its application. Then if $x = [1 \ 0 \ 1 \ 1 \ 1]$, $y = [1 \ 0 \ 0 \ 1 \ 0]$, and $\theta = 1$, what is the result of $\gamma_g(x, y, \theta)$? First, we calculate $\alpha(x, y) = [1 \ 1 \ 2 \ 1 \ 2]$; then

$$\begin{aligned} \alpha(x, y) \bmod 2 &= [1 \ 1 \ 2 \ 1 \ 2] \bmod 2 \\ &= [1 \ 1 \ 0 \ 1 \ 0]; \end{aligned} \quad (11)$$

now

$$\begin{aligned} u_\beta(\alpha(x, y) \bmod 2) &= u_\beta[1 \ 1 \ 0 \ 1 \ 0] \\ &= \sum [\beta(1, 1), \beta(1, 1), \beta(0, 0), \beta(1, 1), \beta(0, 0)] \\ &= 1 + 1 + 0 + 1 + 0 = 3 \end{aligned} \quad (12)$$

and since $m = 5$, $5 - 3 = 2$; given that $2 \leq (\theta = 1)$, then the result of γ_g is 0, and it is decided that given θ the two vectors are not similar. However, if we increase $\theta = 2$, the result of γ_g will be 1, since $2 \leq (\theta = 2)$. In this case, the two vectors will be considered similar. The main idea of the generalized Gamma operator is to indicate (result equal to 1) that two binary vectors are similar allowing up to θ bits to be different and still consider those vectors similar. If more than θ bits are different, the vectors are said to be not similar (result equal to 0). Thus, if $\theta = 0$ both vectors must be equal for γ_g to output a 1.

2.1.1. Gamma Classifier Algorithm. Let $\{(x^\mu, y^\mu) \mid \mu = 1, 2, \dots, p\}$ be the fundamental pattern set with cardinality p ; when a test pattern $\tilde{x} \in \mathbb{R}^n$ with $n \in \mathbb{Z}^+$ is presented to the Gamma classifier, these steps are followed:

- (1) Code the fundamental set using the modified Johnson-Möbius code, obtaining a value e_m for each component of the n -dimensional vectors in the fundamental set. These e_m values are calculated from the original real-valued vectors as

$$e_m(j) = \text{MAX}_{i=1}^p(x_j^i), \quad \forall j \in \{1, 2, \dots, n\}. \quad (13)$$

- (2) Compute the stop parameter $\rho = \text{MAX}_{j=1}^n[e_m(j)]$.
- (3) Code the test pattern using the modified Johnson-Möbius code, with the same parameters used to code the fundamental set. If any e_j (obtained by offsetting, scaling, and truncating the original $\tilde{x}_j$) is greater than the corresponding $e_m(j)$, code it using the larger number of bits.
- (4) Transform the index of all fundamental patterns into two indices, one for their class and another for their position in the class (e.g., x^μ which is the ω th pattern for class i becomes $x^{i\omega}$).
- (5) Initialize θ to 0.
- (6) If $\theta = 0$, test whether $\tilde{x}$ is a fundamental pattern by calculating $\gamma_g(x_j^\mu, \tilde{x}_j, 0)$ for $\mu = 1, 2, \dots, p$ and then computing the initial weighted addition c_μ^0 for each fundamental pattern as follows:

$$c_\mu^0 = \sum_{i=1}^n \gamma_g(x_j^\mu, \tilde{x}_j, 0), \quad \text{for } \mu = 1, 2, \dots, p. \quad (14)$$

If there is a unique maximum, whose value equals n , assign the class associated with such maximum to the test pattern. Consider

$$\tilde{y} = y^\sigma \text{ such that } \text{MAX}_{i=1}^p c_i^0 = c_\sigma^0 = n. \quad (15)$$

- (7) Calculate $\gamma_g(x_j^{i\omega}, \tilde{x}_j, \theta)$ for each component of the fundamental patterns.

- (8) Compute a weighted sum c_i for each class, according to this equation:

$$c_i = \frac{\sum_{\omega=1}^{k_i} \sum_{j=1}^n \gamma_g(x_j^{i\omega}, \tilde{x}_j, \theta)}{k_i}, \quad (16)$$

where k_i is the cardinality in the fundamental set of class i .

- (9) If there is more than one maximum among the different c_i , increment θ by 1 and repeat steps (7) and (8) until there is a unique maximum, or the stop condition $\theta > \rho$ is fulfilled.

- (10) If there is a unique maximum among the c_i , assign $\tilde{x}$ to the class corresponding to such maximum

$$\tilde{y} = y^j \text{ such that } \text{MAX } c_i = c_j. \quad (17)$$

- (11) Otherwise, assign $\tilde{x}$ to the class of the first maximum.

As an example, let us consider the following fundamental patterns:

$$\begin{aligned} x^1 &= \begin{pmatrix} 2 \\ 8 \end{pmatrix}, \\ x^2 &= \begin{pmatrix} 1 \\ 9 \end{pmatrix}, \\ x^3 &= \begin{pmatrix} 6 \\ 3 \end{pmatrix}, \\ x^4 &= \begin{pmatrix} 7 \\ 4 \end{pmatrix} \end{aligned} \quad (18)$$

grouped in two classes: $C_1 = \{x^1, x^2\}$ and $C_2 = \{x^3, x^4\}$. Then the patterns to be classified are

$$\begin{aligned} \tilde{x}^1 &= \begin{pmatrix} 6 \\ 2 \end{pmatrix}, \\ \tilde{x}^2 &= \begin{pmatrix} 3 \\ 9 \end{pmatrix}. \end{aligned} \quad (19)$$

As can be seen, the dimension of all patterns is $n = 2$, and there are 2 classes, both with the same cardinality $k_1 = 2$ and $k_2 = 2$. Now the steps in the algorithm are followed.

- (1) Code the fundamental set with the modified Johnson-Möbius code, obtaining a value e_m for each component. Consider

$$\begin{aligned} x^1 &= \begin{bmatrix} 0000011 \\ 011111111 \end{bmatrix}, \\ x^2 &= \begin{bmatrix} 0000001 \\ 111111111 \end{bmatrix}, \\ x^3 &= \begin{bmatrix} 0111111 \\ 000000111 \end{bmatrix}, \\ x^4 &= \begin{bmatrix} 1111111 \\ 000001111 \end{bmatrix}. \end{aligned} \quad (20)$$

Since the maximum value among the first components of all fundamental patterns is 7, then $e_m(1) = 7$, and the maximum value among the second components is $e_m(2) = 9$. Thus, the binary vectors representing the first components are 7 bits long, while those representing the second component are 9 bits long.

- (2) Compute the stop parameter:

$$\begin{aligned} \rho &= \text{MAX}_{j=1}^n [e_m(j)] = \text{MAX}_{j=1}^2 [e_m(1), e_m(2)] \\ &= \text{MAX}(7, 9) = 7. \end{aligned} \quad (21)$$

- (3) Code the patterns to be classified into the modified Johnson-Möbius code, using the same parameters used with the fundamental set:

$$\begin{aligned} \tilde{x}^1 &= \begin{bmatrix} 0111111 \\ 000000011 \end{bmatrix}, \\ \tilde{x}^2 &= \begin{bmatrix} 0000111 \\ 111111111 \end{bmatrix}. \end{aligned} \quad (22)$$

Again, the binary vectors representing the first components are 7 bits long, while those representing the second component are 9 bits long.

- (4) Transform the index of all fundamental patterns into two indices, one for the class they belong to and another for their position in the class. Consider

$$\begin{aligned} x^1 &= x^{11}, \\ x^2 &= x^{12}, \\ x^3 &= x^{21}, \\ x^4 &= x^{22}. \end{aligned} \quad (23)$$

Given that $C_1 = \{x^1, x^2\}$ and $C_2 = \{x^3, x^4\}$, we know that both x^1 and x^2 belong to class C_1 , so the first index for both patterns becomes 1; the second index is used to differentiate between patterns assigned to the same class; thus x^1 becomes x^{11} , and x^2 is now x^{12} . Something similar happens to x^3 and x^4 , but with the first index being 2 since they belong to class C_2 : x^3 becomes x^{21} , and x^4 is now x^{22} .

- (5) Initialize = 0.

- (6) Calculate $\gamma_g(x_j^{i\omega}, \tilde{x}_j, \theta)$ for each component of the fundamental patterns in each class. Thus, for $\tilde{x}^1$ we have

$$\begin{aligned} \gamma_g(x_1^{11}, \tilde{x}_1^1, 0) &= 0, \\ \gamma_g(x_2^{11}, \tilde{x}_2^1, 0) &= 0, \\ \gamma_g(x_1^{12}, \tilde{x}_1^1, 0) &= 0, \\ \gamma_g(x_2^{12}, \tilde{x}_2^1, 0) &= 0, \\ \gamma_g(x_1^{21}, \tilde{x}_1^1, 0) &= 1, \\ \gamma_g(x_2^{21}, \tilde{x}_2^1, 0) &= 0, \\ \gamma_g(x_1^{22}, \tilde{x}_1^1, 0) &= 0, \\ \gamma_g(x_2^{22}, \tilde{x}_2^1, 0) &= 0. \end{aligned} \quad (24)$$

Since $\theta = 0$, for γ_g to give a result of 1 it is necessary that both vectors $x_j^{i\omega}$ and $\tilde{x}_j^1$ are equal. But only one fundamental pattern has its first component equal to that of $\tilde{x}^1$; $x_1^{21} = [0111111] = \tilde{x}_1^1$, while no fundamental pattern has a second component equal to that of the test pattern. Given that this is the only case for which a component of a fundamental pattern is equal to the same component of the test pattern, it is also the only instance in which γ_g outputs 1.

Now, for $\tilde{x}^2$ we have

$$\begin{aligned} \gamma_g(x_1^{11}, \tilde{x}_1^2, 0) &= 0, \\ \gamma_g(x_2^{11}, \tilde{x}_2^2, 0) &= 0, \\ \gamma_g(x_1^{12}, \tilde{x}_1^2, 0) &= 0, \\ \gamma_g(x_2^{12}, \tilde{x}_2^2, 0) &= 1, \\ \gamma_g(x_1^{21}, \tilde{x}_1^2, 0) &= 0, \\ \gamma_g(x_2^{21}, \tilde{x}_2^2, 0) &= 0, \\ \gamma_g(x_1^{22}, \tilde{x}_1^2, 0) &= 0, \\ \gamma_g(x_2^{22}, \tilde{x}_2^2, 0) &= 0. \end{aligned} \quad (25)$$

Again $\theta = 0$, thus forcing both vectors $x_j^{1\omega}$ and $\tilde{x}_j^2$ to be equal in order to obtain a 1. Similarly to what happened in the case of $\tilde{x}^1$, there is but one instance of such occurrence: $x_2^{12} = [111111] = \tilde{x}_2^2$.

(7) Compute a weighted sum c_i for each class:

$$\begin{aligned} c_1 &= \frac{\sum_{\omega=1}^{k_1} \sum_{j=1}^n \gamma_g(x_j^{1\omega}, \tilde{x}_j^1, \theta)}{k_1} \\ &= \frac{\sum_{\omega=1}^2 \sum_{j=1}^2 \gamma_g(x_j^{1\omega}, \tilde{x}_j^1, \theta)}{2} \\ &= \frac{\sum_{\omega=1}^2 [(0+0), (0+0)]}{2} = \frac{(0+0)}{2} = 0. \end{aligned} \quad (26)$$

Here we add together the results obtained on all components of all fundamental patterns belonging to class C_1 . Since all gave 0 (i.e., none of these were similar to the corresponding component of the test pattern given the value of θ in effect), the result of the weighted addition for class C_1 is $c_1 = 0$. One has

$$\begin{aligned} c_2 &= \frac{\sum_{\omega=1}^{k_2} \sum_{j=1}^n \gamma_g(x_j^{2\omega}, \tilde{x}_j^1, \theta)}{k_2} \\ &= \frac{\sum_{\omega=1}^2 \sum_{j=1}^2 \gamma_g(x_j^{2\omega}, \tilde{x}_j^1, \theta)}{2} \\ &= \frac{\sum_{\omega=1}^2 [(1+0), (0+0)]}{2} = \frac{(1+0)}{2} = 0.5. \end{aligned} \quad (27)$$

In this case, there was one instance similar to the corresponding component of the test pattern (equal, since $\theta = 0$ during this run). Thus the weighted addition for class C_2 is $c_2 = 0.5$, given that the sum of results is divided by the cardinality of the class, $k = 2$. And for $\tilde{x}^2$,

$$\begin{aligned} c_1 &= \frac{\sum_{\omega=1}^{k_1} \sum_{j=1}^n \gamma_g(x_j^{1\omega}, \tilde{x}_j^2, \theta)}{k_1} \\ &= \frac{\sum_{\omega=1}^2 \sum_{j=1}^2 \gamma_g(x_j^{1\omega}, \tilde{x}_j^2, \theta)}{2} \\ &= \frac{\sum_{\omega=1}^2 [(0+0), (0+1)]}{2} = \frac{(0+0)}{2} = 0.5. \end{aligned} \quad (28)$$

There was one result of γ_g equal to 1, which coupled with the cardinality of class C_1 being 2 makes $c_1 = 0.5$. One has

$$\begin{aligned} c_2 &= \frac{\sum_{\omega=1}^{k_2} \sum_{j=1}^n \gamma_g(x_j^{2\omega}, \tilde{x}_j^2, \theta)}{k_2} \\ &= \frac{\sum_{\omega=1}^2 \sum_{j=1}^2 \gamma_g(x_j^{2\omega}, \tilde{x}_j^2, \theta)}{2} \\ &= \frac{\sum_{\omega=1}^2 [(0+0), (0+0)]}{2} = \frac{(0+0)}{2} = 0. \end{aligned} \quad (29)$$

No similarities were found between the fundamental patterns of class C_2 and the test pattern $\tilde{x}^2$ (for $\theta = 0$); thus $c_2 = 0$.

- (8) If there is more than one maximum among the different c_i , increment θ by 1 and repeat steps (6) and (7) until there is a unique maximum, or the stop condition $\theta \geq \rho$ is fulfilled. There is a unique maximum (for each test pattern), so we go directly to step (9).
- (9) If there is a unique maximum, assign y to the class corresponding to such maximum. For $\tilde{x}^1$,

$$C_{\tilde{x}^1} = C_2 \text{ since} \quad (30)$$

$$\text{MAX}_{i=1}^2 c_i = \text{MAX}(0, 0.5) = 0.5 = c_2,$$

while for $\tilde{x}^2$

$$C_{\tilde{x}^2} = C_1 \text{ since} \quad (31)$$

$$\text{MAX}_{i=1}^2 c_i = \text{MAX}(0.5, 0) = 0.5 = c_1.$$

- (10) Otherwise, assign y to the class of the first maximum. This is unnecessary since both test patterns have already been assigned a class.

2.2. Sliding Windows. A widespread approach for data stream mining is the use of a sliding window to keep only a representative portion of the data. We are not interested in keeping all the information of the data stream, particularly when we have to meet the memory space constraint required for algorithms working in this context. This technique is able to deal with concept drift, eliminating those data points that come from an old concept. According to [5], the training window size can be fixed or variable over time.

Sliding windows of a fixed size store in memory a fixed number of the w most recent examples. Whenever a new example arrives, it is saved to memory and the oldest one is discarded. These types of windows are similar to first-in first-out data structures. This simple adaptive learning method is often used as a baseline in evaluation of new algorithms.

Sliding windows of variable size vary the number of examples in a window over time, typically depending on the indications of a change detector. A straightforward approach is to shrink the window whenever a change is detected, such that the training data reflects the most recent concept, and grow the window otherwise.

2.3. Proposed Method. The Gamma classifier has previously been used for time series prediction in air pollution [37] and oil well production [40], showing very promising results. The current work takes the Gamma classifier and combines it with a sliding window approach, with the aim of developing a new method capable of performing classification over a continuous data stream, referred to as *DS-Gamma classifier*.

```

(1) Initialize window
(2) for each  $x^\tau$  do
(3)   If  $\tau < W$  then
(4)     Add  $x^\tau$  to the tail of the window
(5)   Else
(6)     Use  $x^\tau$  to classify
(7)     Add  $x^\tau$  to the tail of the window

```

ALGORITHM 1: Learning policy algorithm.

A data stream S can be defined as a sequence of data elements of the form $S = \{s_1, s_2, \dots, s_{\text{now}}\}$ from a universe of size D , where D is huge and potentially infinite but countable. We assume that each element s_i has the following form:

$$s_i = (x^\tau, y^\tau), \quad (32)$$

where x^τ is an input vector of size n and y^τ is the class label associated with it. $\tau \in \mathbb{Z}^+$ is an index to indicate the sequential order of the elements in the data stream. τ will be bounded by the size of the data stream such that if the data stream is infinite τ will take infinite values, but countable.

One of the main challenges for mining data streams is the ability to maintain an accurate decision model. The current model has to incorporate new knowledge from new data and also it should forget old information when data is outdated. To address these issues the proposed method implements a learning and forgetting policy. The learning (insertion)/forgetting (removal) policies were designed to guarantee the space and temporal relevance of the data and the consistency of the current model.

The proposed method implements a fixed window approach. This approach achieves concept drift detection in an indirect way by considering only the most recent data. Algorithm 1 outlines the general steps used as learning policy. Initially, the window is created with the first W labeled records, where W is the size of the window. After that every sample is used for classification and evaluation of the performance, and then it is used to update the window.

Classifiers that deal with concept drifts are forced to implement forgetting, adaptation, or drift detection mechanisms in order to adjust to changing environments. In this proposal a forgetting mechanism is implemented. Algorithm 2 outlines the basic forgetting policy implemented in this proposal. The oldest element in the window x^δ is removed.

2.4. MOA. Massive Online Analysis (MOA) is a software environment for implementing algorithms and running experiments for online learning from evolving data streams [41]. MOA contains a collection of offline and online algorithms for both classification and clustering as well as tools for evaluation on data streams with concept drifts. MOA also can be used to build synthetic data streams using generators, reading ARFF files, or joining several streams. MOA streams generators allow simulating potentially infinite sequences of

TABLE 2: Datasets characteristics.

Dataset	Instances	Attributes	Classes
Agrawal	100,000	10	2
Bank	45211	16	2
CoverType	581,012	54	7
Electricity	45,312	8	2
Hyperplane	100,000	10	2
RandomRBF	100,000	10	2

data with different characteristics (different types of concept drifts and recurrent concepts, among others). All the algorithms used for comparative analysis are implemented under the MOA framework.

2.5. Data Streams. This section provides a brief description of the data streams used during the experimental phase. The proposed solution has been tested using synthetic and real data streams. For data stream classification problems, just a few suitable data streams can be successfully used for evaluation. A few data streams with enough examples are hosted in [42, 43]. For further evaluation, the use of synthetic data streams generators becomes necessary. The synthetic data streams used in our experiments were generated using the MOA framework mentioned in the previous section. A summary of the main characteristics of the data streams is shown in Table 2. Descriptions of the data streams were taken from [42, 43] in the case of real ones and from [44] for the synthetics.

2.5.1. Real World Data Streams

Bank Marketing Dataset. The Bank Marketing dataset is available from UCI Machine Learning Repository [43]. This data is related to direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe or not a term deposit. It contains 45,211 instances. Each instance has 16 attributes and the class label. The dataset has two classes that identified if a bank term deposit would be or would not be subscribed.

Electricity Dataset. This data was collected from the Australian New South Wales electricity market. In this market, prices are not fixed and are affected by demand and supply of the market. The prices of the market are set every five minutes. The dataset was first described by Harries in [45]. During the time period, described in the dataset, the electricity market was expanded with the inclusion of adjacent areas, which leads to more elaborate management of the supply. The excess production of one region could be sold on the adjacent region. The Electricity dataset contains 45,312 instances. Each example of the dataset refers to a period of 30 minutes (i.e., there are 48 instances for each time period of one day). Each example on the dataset has 8 fields: the day of week, the time stamp, the New South Wales electricity demand, the New South Wales electricity price, the Victoria electricity demand, the Victoria electricity price, the scheduled electricity transfer


```

(1) for each  $x^\tau$  do
(2)   If  $\tau > W$  then
(3)     Remove  $x^\delta$  ( $\delta = \tau - W + 1$ ) from the beginning of the window

```

ALGORITHM 2: Forgetting policy algorithm.

between states, and the class label. The class label identifies the change of the price relative to a moving average of the last 24 hours. We use the normalized versions of these datasets so that the numerical values are between 0 and 1. The dataset is available from [42].

Forest CoverType Dataset. The Forest CoverType dataset is one of the largest datasets available from the UCI Machine Learning Repository [43]. This dataset contains information that describes forest cover types from cartographic variables. A given observation (30×30 meter cell) was determined from the US Forest Service (USFS) Region 2 Resource Information System (RIS) data. It contains 581,012 instances and 54 attributes. Data is in raw form (not scaled) and contains binary (0 or 1) columns of data for qualitative independent variables (wilderness areas and soil types). The dataset has 7 classes that identify the type of forest cover.

2.5.2. Synthetic Data Streams

Agrawal Data Stream. This generator was originally introduced by Agrawal et al. in [46]. The MOA implementation was used to generate the data streams for the experimental phase in this work. The generator produces a stream containing nine attributes. Although not explicitly stated by the authors, a sensible conclusion is that these attributes describe hypothetical loan applications. There are ten functions defined for generating binary class labels from the attributes. Presumably these determine whether the loan should be approved. Perturbation shifts numeric attributes from their true value, adding an offset drawn randomly from a uniform distribution, the range of which is a specified percentage of the total value range [44]. For each experiment, a data stream with 100,000 examples was generated.

RandomRBF Data Stream. This generator was devised to offer an alternate complex concept type that is not straightforward to approximate with a decision tree model. The RBF (Radial Basis Function) generator works as follows. A fixed number of random centroids are generated. Each center has a random position, a single standard deviation, class label, and weight. New examples are generated by selecting a center at random, taking weights into consideration so that centers with higher weight are more likely to be chosen. A random direction is chosen to offset the attribute values from the central point. The length of the displacement is randomly drawn from a Gaussian distribution with standard deviation determined by the chosen centroid. The chosen centroid also determines the class label of the example. Only numeric attributes are generated [44]. For our experiments, a data stream with

100,000 data instances, 10 numerical attributes, and 2 classes was generated.

Rotating Hyperplane Data Stream. This synthetic data stream was generated using the MOA framework. A hyperplane in d -dimensional space is the set of points x that satisfy [44]

$$\sum_{i=1}^d x_i w_i = w_0 = \sum_{i=1}^d w_i, \quad (33)$$

where x_i is the i th coordinate of x , examples for which $\sum_{i=1}^d x_i w_i \geq w_0$ are labeled positive and examples for which $\sum_{i=1}^d x_i w_i < w_0$ are labeled negative. Hyperplanes are useful for simulating time-changing concepts, because orientation and position of the hyperplane can be adjusted in a gradual way if we change the relative size of the weights. In MOA, they introduce change to this data stream by adding drift to each weight attribute using this formula: $w_i = w_i + d\sigma$, where σ is the probability that the direction of change is reversed and d is the change applied to every example. For our experiments, a data stream with 100,000 data instances, 10 numerical attributes, and 2 classes was generated.

2.6. Algorithms Evaluation. One of the objectives of this work is to perform a consistent comparison between the classification performance of our proposal and the classification performance of other well-known data stream pattern classification algorithms. Evaluation of data stream algorithms is a relatively new field that has not been studied as well as evaluation on batch learning. In traditional batch learning evaluation has been mainly limited to evaluating a model using different random reorder of the same static dataset (k -fold cross validation, leave-one-out, and bootstrap) [2]. For data stream scenarios the problem of infinite data raises new challenges. On the other hand, the batch approaches cannot effectively measure accuracy of a model in a context with concepts that change over time. In the data stream setting, one of the main concerns is how to design an evaluation practice that allows us to obtain a reliable measure of accuracy over time. According to [41], two main approaches arise.

2.6.1. Holdout. When traditional batch learning reaches a scale where cross validation is too time-consuming, it is often accepted to instead measure performance on a single holdout set. An extension of this approach can be applied for data stream scenarios. First a set of M instances is used to train the model; then a set of the following unseen N instances is used to test the model; then it is again trained with the next M instances and tested with the subsequent N instances and so forth.

2.6.2. Interleaved Test-Then-Train or Prequential. Each individual example can be used to test the model before it is used for training, and from this the accuracy can be incrementally updated. When intentionally performed in this order, the model is always being tested on examples it has not seen.

Holdout evaluation gives a more accurate estimation of the performance of the classifier on more recent data. However, it requires recent test data that is sometimes difficult to obtain, while the prequential evaluation has the advantage that no holdout set is needed for testing, making maximum use of the available data. In [47], Gama et al. present a general framework to assess data stream algorithms. They studied different mechanisms of evaluation that include holdout estimator, the prequential error, and the prequential error estimated over a sliding window or using fading factors. They stated that the use of prequential error with forgetting mechanisms reveals to be advantageous in assessing performance and in comparing stream learning algorithms. Since our proposed method is working with a sliding window approach, we have selected the prequential error over sliding window for evaluation of the proposed method. The prequential error P_W for a sliding window of size W computed at time i is based on an accumulated sum of a 0-1 loss function L between the predicted $\hat{y}_k$ values and the observed values y_k using the following expression [47]:

$$P_W(i) = \frac{1}{W} \sum_{k=i-W+1}^i L(y_k, \hat{y}_k). \quad (34)$$

We also evaluate the performance results of each competing algorithm with the Gamma classifier using a statistical test. We use the Wilcoxon test [48] to assess the statistical significance of the results. The Wilcoxon signed-ranks test is a nonparametric test, which can be used to rank the differences in performances of two classifiers for each dataset, ignoring the signs and comparing the ranks for the positive and the negative differences.

2.7. Experimental Setup. To ensure valid comparison of classification performance, the same conditions and validation schemes were applied in each experiment. Classification performance of each of the compared algorithms was calculated using the prequential error approach. These results will be used to compare the performance of our proposal and other data stream classification algorithms. We have aimed at choosing representative algorithms from different approaches covering the state of the art in data stream classification, such as Hoeffding Tree, IBL, Naïve Bayes with DDM (Drift Detection Method), SVM, and ensemble methods. All of these algorithms are implemented in the MOA framework. Further details on the implementation of these algorithms can be found in [44]. For all data streams, with exception of the Electricity dataset, the experiments were executed 10 times and the average of the performance and execution time results are presented. Records in the Electricity dataset have a temporal relation that should not be altered; for this reason with this specific dataset the experiment was executed just one time. The Bank and CoverType datasets

were reordered 10 times using the randomized filter from the WEKA platform [49] for each experiment. For the synthetic data streams (Agrawal, Rotating Hyperplane, and RandomRBF), 10 data streams were generated for each one of them using different random seeds in MOA. Performance was calculated using the prequential error introduced in the previous section. We also measure the total time used for classification to evaluate the efficiency of each algorithm. All experiments were conducted using a personal computer with an Intel Core i3-2100 processor running Ubuntu 13.04 64-bit operating system with 4096 GB of RAM.

3. Results and Discussion

In this section, we present and discuss the results obtained during the experimental phase, throughout which six data streams, three real and three synthetic, were used to obtain the classification and time performance for each of the compared classification algorithms. First, we evaluate the sensibility of the proposed method to the change on the windows size, which is a parameter that can largely influence the performance of the classifier. Table 3 shows the average accuracy of the DS-Gamma classifier for different window sizes; best accuracy for each data stream is emphasized with boldface. Table 4 shows the average total time used to classify each data stream. The results of the average performance with indicator of the standard deviation are also depicted in Figure 1.

In general, the DS-Gamma classifier performance improved as the window size increases with the exception of Electricity data stream, for this data stream performance is better with smaller window sizes. It is worth noting that the DS-Gamma classifier achieved its best performance for the Electricity data stream with a window size of 50 with a performance of 89.12%. This performance is better than all the performances achieved by all the other compared algorithms with a window size = 1000. For the other data streams, the best performance was generally obtained with window sizes between 500 and 1000. With window sizes greater than 1000 performance keeps stable and in some cases even degrades, as we can observe in the results from the Bank and Hyperplane data streams in Figure 1. Figure 2 shows the average classification time for different window size. A linear trend line (dotted line) and the coefficient of determination R^2 were included in the graphic of each data stream. The R^2 values ranging between 0.9743 and 0.9866 show that time is growing in linear way as the size of the window increases.

To evaluate the DS-Gamma classifier, we compared its performance with other data stream classifiers. Table 5 presents the performance results of each evaluated algorithm on each data stream, including the standard deviation. Best performance is emphasized with boldface. We also include box plot with the same results in Figure 3. For the Agrawal data stream Hoeffding Tree and the DS-Gamma classifier achieved the two best performances. OzaBoostAdwin presents the best performance for the CoverType and Electricity data streams, but this ensemble classifier exhibits the highest classification time for all the data streams, as we can observe in Table 6 and

TABLE 3: Average accuracy of DS-Gamma for different windows size.

Dataset	Window size									
	50	100	200	500	750	1000	1250	1500	1750	2000
Agrawal	79.64	85.97	89.86	92.59	92.88	92.82	92.67	92.51	92.32	92.07
Bank	68.19	73.40	77.05	79.13	79.12	78.96	78.60	78.24	77.76	77.37
CoverType	38.21	37.61	39.70	44.30	46.59	47.77	48.62	49.19	49.51	49.74
Electricity	89.12	85.26	81.71	79.84	78.58	77.14	76.36	75.94	75.31	74.49
Hyperplane	78.25	82.65	85.28	86.56	86.55	86.30	85.98	85.65	85.34	85.00
RandomRBF	64.61	67.39	69.33	70.56	70.81	70.85	70.74	70.63	70.50	70.33

TABLE 4: Averaged classification time (seconds) of DS-Gamma for different windows size.

Dataset	Window size									
	50	100	200	500	750	1000	1250	1500	1750	2000
Agrawal	2.24	4.50	7.93	19.26	28.04	39.00	45.92	57.57	63.90	73.49
Bank	0.66	1.14	2.07	4.75	7.01	9.17	11.54	14.55	15.83	17.87
CoverType	83.47	140.81	251.20	340.05	254.12	335.03	428.94	522.39	669.24	676.99
Electricity	0.34	0.53	0.98	2.33	3.56	4.63	5.99	7.05	8.30	9.49
Hyperplane	2.24	4.01	7.56	18.19	26.76	30.28	44.28	52.96	61.87	70.63
RandomRBF	2.79	4.09	7.73	18.54	27.35	30.72	45.02	54.12	62.23	71.04

TABLE 5: Averaged accuracy (%) comparison.

Classifier	Agrawal		Bank		CoverType		Electricity		Hyperplane		RandomRBF	
	Acc.	σ	Acc.	σ	Acc.	σ	Acc.	σ	Acc.	σ	Acc.	σ
Bayes-DDM	88.16	0.19	83.38	1.41	67.61	6.81	81.05	—	86.02	1.39	71.87	3.82
Perceptron DDM	67.20	0.11	73.43	10.53	70.76	5.55	78.61	—	85.65	1.49	71.88	5.93
Hoeffding Tree	94.39	0.24	88.52	0.11	72.29	2.68	82.41	—	82.41	5.62	86.72	1.69
Active Classifier	90.82	1.62	88.21	0.14	67.31	1.03	78.59	—	79.50	6.55	75.32	2.70
IBL-DS	86.76	0.48	86.52	0.21	61.73	9.46	75.34	—	84.20	0.77	93.32	1.21
AWE	92.17	0.59	87.90	0.38	76.16	3.66	78.36	—	85.88	1.65	91.98	1.14
OzaBoostAdwin	87.11	2.62	87.67	0.50	78.55	4.97	87.92	—	83.21	1.91	91.49	1.14
SGD	67.20	0.11	76.56	1.66	59.01	7.37	84.52	—	83.19	0.42	61.74	3.31
SPegasos	55.82	0.17	77.46	1.99	50.47	5.17	57.60	—	52.80	6.08	53.33	3.13
DS-Gamma	92.82	0.10	78.96	0.22	47.77	4.31	77.14	—	86.30	1.35	70.85	2.54

TABLE 6: Averaged classification time (seconds) comparison.

Classifier	Agrawal		Bank		CoverType		Electricity		Hyperplane		RandomRBF	
	Time	σ	Time	σ	Time	σ	Time	σ	Time	σ	Time	σ
Bayes-DDM	0.62	0.11	0.39	0.08	17.76	1.62	0.61	—	0.70	0.01	0.69	0.01
Perceptron DDM	0.40	0.02	0.20	0.05	8.28	0.10	0.19	—	0.45	0.02	0.44	0.02
Hoeffding Tree	0.98	0.23	0.54	0.03	30.05	11.06	0.78	—	1.36	0.05	1.40	0.05
Active Classifier	0.64	0.03	0.26	0.02	15.14	0.74	0.23	—	0.72	0.04	0.70	0.02
IBL-DS	53.19	2.11	38.51	8.26	1057.60	60.95	8.61	—	86.39	0.96	88.10	1.06
AWE	14.32	0.20	6.63	0.30	1012.47	304.41	6.70	—	18.03	3.64	37.52	3.03
OzaBoostAdwin	192.62	44.47	46.93	10.56	10589.73	3896.64	39.75	—	223.99	51.34	57.04	16.66
SGD	0.14	0.04	0.11	0.009	4.035	0.07	0.12	—	0.27	0.02	0.27	0.02
SPegasos	0.22	0.006	0.11	0.016	1.684	0.12	0.12	—	0.31	0.05	0.24	0.01
DS-Gamma	39.00	3.58	9.16	0.05	335.03	12.91	4.63	—	30.28	0.79	30.72	0.94

FIGURE 1: Average accuracy of DS-Gamma for different windows size.

Figure 4. This can be a serious disadvantage, especially in data stream scenarios. The lowest times are achieved by the perceptron with Drift Detection Method (DDM), but its performance is generally low when compared to the other methods. A fact that caught our attention was the low performance of the DS-Gamma classifier for the CoverType data stream. This data stream presents a heavy class imbalance that seems to be negatively affecting the performance of the classifier. As we can observe in step (8) of the DS-Gamma

classifier algorithm, classification relies on a weighted sum per class; when a class greatly outnumbers the other(s), this sum will be biased to the class with most members.

To statistically compare the results obtained by the DS-Gamma classifier and the other evaluated algorithms the Wilcoxon signed-ranks test was used. Table 7 shows the values of the asymptotic significance (2-sided) obtained by the Wilcoxon test using $\alpha = 0.05$. The obtained values are all greater than α , so the null hypothesis is not rejected and

FIGURE 2: Averaged classification time (seconds) of DS-Gamma for different windows size.

we can infer that there are no significant differences among the compared algorithms.

4. Conclusions

In this paper, we describe a novel method for the task of pattern classification over a continuous data stream based on an associative model. The proposed method is a supervised pattern recognition model based on the Gamma classifier. During the experimental phase, the proposed method was tested using different data stream scenarios with real and synthetic data streams. The proposed method presented very promising results in terms of performance and classification

TABLE 7: Wilcoxon results.

Compared algorithms	Asymptotic significance (2-sided)
Gamma-Bayes-DDM	0.345
Gamma-Perceptron DDM	0.917
Gamma-Hoeffding Tree	0.116
Gamma-Active Classifier	0.463
Gamma-IBL-DS	0.345
Gamma-AWE	0.116
Gamma-OzaBoostAdwin	0.116
Gamma-SGD	0.600
Gamma-SPegasos	0.075

FIGURE 3: Averaged accuracy (%) comparison.

time when compared with some of the state-of-the-art algorithms for data stream classification.

We also study the effect of the window size over the classification accuracy and time. In general, accuracy improved as the window size increased. We observed that best performances were obtained with window sizes between 500 and

1000. For larger window sizes performance remained stable and in some cases even declined.

Since the proposed method based its classification on a weighted sum per class, it is strongly affected by severe class imbalance, as we can observe from the results obtained with the experiments performed with the CoverType dataset,

FIGURE 4: Averaged classification time (seconds) comparison.

a heavily imbalance dataset, where the accuracy of the classifier was the worst of the compared algorithms. As future work a mechanism to address severe class imbalance should be integrated to improve classification accuracy.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank the Instituto Politécnico Nacional (Secretaría Académica, COFAA, SIP, CIC, and CIDETEC), the CONACyT, and SNI for their economical support to develop this work. The support of the University of Porto was provided during the research stay from August to December, 2014. The authors also want to thank Dr. Yenny Villuendas Rey for her contribution to perform the statistical significance analysis.

References

- [1] V. Turner, J. F. Gantz, D. Reinsel, and S. Minton, "The digital universe of opportunities: rich data and the increasing value of the internet of things," Tech. Rep., IDC Information and Data, 2014, <http://www.emc.com/leadership/digital-universe/2014view/index.htm>.
- [2] A. Bifet, *Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*, IOS Press, Amsterdam, The Netherlands, 2010.
- [3] J. Gama, *Knowledge Discovery from Data Streams*, Chapman & Hall/CRC, 2010.
- [4] J. Beringer and E. Hüllermeier, "Efficient instance-based learning on data streams," *Intelligent Data Analysis*, vol. 11, no. 6, pp. 627–650, 2007.
- [5] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, no. 4, article 44, 2014.
- [6] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 71–80, Boston, Mass, USA, August 2000.
- [7] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*, pp. 97–106, San Francisco, Calif, USA, August 2001.
- [8] C. Liang, Y. Zhang, P. Shi, and Z. Hu, "Learning very fast decision tree from uncertain data streams with positive and unlabeled samples," *Information Sciences*, vol. 213, pp. 50–67, 2012.
- [9] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda, "Decision trees for mining data streams based on the gaussian approximation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 1, pp. 108–119, 2014.
- [10] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine Learning*, vol. 23, no. 1, pp. 69–101, 1996.
- [11] F. Ferrer-Troyano, J. S. Aguilar-Ruiz, and J. C. Riquelme, "Data streams classification by incremental rule learning with parameterized generalization," in *Proceedings of the 2006 ACM Symposium on Applied Computing*, pp. 657–661, Dijon, France, April 2006.
- [12] D. Mena-Torres and J. S. Aguilar-Ruiz, "A similarity-based approach for data stream classification," *Expert Systems with Applications*, vol. 41, no. 9, pp. 4224–4234, 2014.
- [13] A. Shaker and E. Hüllermeier, "IBLStreams: a system for instance-based classification and regression on data streams," *Evolving Systems*, vol. 3, no. 4, pp. 235–249, 2012.
- [14] A. Ghazikhani, R. Monsefi, and H. S. Yazdi, "Recursive least square perceptron model for non-stationary and imbalanced data stream classification," *Evolving Systems*, vol. 4, no. 2, pp. 119–131, 2013.
- [15] A. Ghazikhani, R. Monsefi, and H. S. Yazdi, "Ensemble of online neural networks for non-stationary and imbalanced data streams," *Neurocomputing*, vol. 122, pp. 535–544, 2013.
- [16] A. Sancho-Asensio, A. Orriols-Puig, and E. Golobardes, "Robust on-line neural learning classifier system for data stream classification tasks," *Soft Computing*, vol. 18, no. 8, pp. 1441–1461, 2014.
- [17] D. M. Farid, L. Zhang, A. Hossain et al., "An adaptive ensemble classifier for mining concept drifting data streams," *Expert Systems with Applications*, vol. 40, no. 15, pp. 5895–5906, 2013.
- [18] D. Brzezinski and J. Stefanowski, "Combining block-based and online methods in learning ensembles from concept drifting data streams," *Information Sciences*, vol. 265, pp. 50–67, 2014.
- [19] G. Ditzler and R. Polikar, "Incremental learning of concept drift from streaming imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 10, pp. 2283–2301, 2013.
- [20] M. J. Hosseini, Z. Ahmadi, and H. Beigy, "Using a classifier pool in accuracy based tracking of recurring concepts in data stream classification," *Evolving Systems*, vol. 4, no. 1, pp. 43–60, 2013.
- [21] N. Saunier and S. Midenet, "Creating ensemble classifiers through order and incremental data selection in a stream," *Pattern Analysis and Applications*, vol. 16, no. 3, pp. 333–347, 2013.
- [22] M. M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 6, pp. 859–874, 2011.
- [23] A. Alzghoul, M. Löfstrand, and B. Backe, "Data stream forecasting for system fault prediction," *Computers and Industrial Engineering*, vol. 62, no. 4, pp. 972–978, 2012.
- [24] J. Kranjc, J. Smailović, V. Podpečan, M. Grčar, M. Žnidaršič, and N. Lavrač, "Active learning for sentiment analysis on data streams: methodology and workflow implementation in the ClowdFlows platform," *Information Processing and Management*, vol. 51, no. 2, pp. 187–203, 2015.
- [25] J. Smailović, M. Grčar, N. Lavrač, and M. Žnidaršič, "Stream-based active learning for sentiment analysis in the financial domain," *Information Sciences*, vol. 285, pp. 181–203, 2014.
- [26] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter, "Pegasos: primal estimated sub-gradient solver for SVM," in *Proceedings of the 24th International Conference on Machine Learning*, pp. 807–814, June 2007.
- [27] Z. Miller, B. Dickinson, W. Deitrick, W. Hu, and A. H. Wang, "Twitter spammer detection using data stream clustering," *Information Sciences*, vol. 260, pp. 64–73, 2014.
- [28] B. Krawczyk and M. Woźniak, "Incremental weighted one-class classifier for mining stationary data streams," *Journal of Computational Science*, vol. 9, pp. 19–25, 2015.
- [29] J. Gama, "A survey on learning from data streams: current and future trends," *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 45–55, 2012.
- [30] L. I. Kuncheva, "Change detection in streaming multivariate data using likelihood detectors," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 5, pp. 1175–1180, 2013.
- [31] G. J. Ross, N. M. Adams, D. K. Tasoulis, and D. J. Hand, "Exponentially weighted moving average charts for detecting concept drift," *Pattern Recognition Letters*, vol. 33, no. 2, pp. 191–198, 2012.
- [32] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence—SBIA 2004*, vol. 3171 of *Lecture Notes in Computer Science*, pp. 286–295, Springer, Berlin, Germany, 2004.
- [33] X. Wu, P. Li, and X. Hu, "Learning from concept drifting data streams with unlabeled data," *Neurocomputing*, vol. 92, pp. 145–155, 2012.
- [34] X. Qin, Y. Zhang, C. Li, and X. Li, "Learning from data streams with only positive and unlabeled data," *Journal of Intelligent Information Systems*, vol. 40, no. 3, pp. 405–430, 2013.

- [35] M. M. Masud, C. Woolam, J. Gao et al., "Facing the reality of data stream classification: coping with scarcity of labeled data," *Knowledge and Information Systems*, vol. 33, no. 1, pp. 213–244, 2012.
- [36] J. Read, A. Bifet, G. Holmes, and B. Pfahringer, "Scalable and efficient multi-label classification for evolving data streams," *Machine Learning*, vol. 88, no. 1-2, pp. 243–272, 2012.
- [37] I. López-Yáñez, A. J. Argüelles-Cruz, O. Camacho-Nieto, and C. Yáñez-Márquez, "Pollutants time-series prediction using the gamma classifier," *International Journal of Computational Intelligence Systems*, vol. 4, no. 4, pp. 680–711, 2011.
- [38] M. E. Acevedo-Mosqueda, C. Yáñez-Márquez, and I. López-Yáñez, "Alpha-beta bidirectional associative memories: theory and applications," *Neural Processing Letters*, vol. 26, no. 1, pp. 1–40, 2007.
- [39] C. Yáñez-Márquez, E. M. Felipe-Riverón, I. López-Yáñez, and R. Flores-Carapia, "A novel approach to automatic color matching," in *Progress in Pattern Recognition, Image Analysis and Applications*, vol. 4225 of *Lecture Notes in Computer Science*, pp. 529–538, Springer, Berlin, Germany, 2006.
- [40] I. López-Yáñez, L. Sheremetov, and C. Yáñez-Márquez, "A novel associative model for time series data mining," *Pattern Recognition Letters*, vol. 41, no. 1, pp. 23–33, 2014.
- [41] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Data stream mining: a practical approach," Tech. Rep., University of Waikato, 2011.
- [42] Massive Online Analysis (MOA), 2014, <http://moa.cms.waikato.ac.nz/datasets/>.
- [43] K. Bache and M. Lichman, *UCI Machine Learning Repository*, School of Information and Computer Science, University of California, Irvine, Calif, USA, 2013, <http://archive.ics.uci.edu/ml>.
- [44] A. Bifet, R. Kirkby, P. Kranen, and P. Reutemann, "Massive Online Analysis Manual," 2012, <http://moa.cms.waikato.ac.nz/documentation/>.
- [45] M. Harries, "Splice-2 comparative evaluation: electricity pricing," Tech. Rep., University of South Wales, Cardiff, UK, 1999.
- [46] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: a performance perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, pp. 914–925, 1993.
- [47] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Machine Learning*, vol. 90, no. 3, pp. 317–346, 2013.
- [48] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.
- [49] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, *WEKA 3: Data Mining Software in Java*, 2010, <http://www.cs.waikato.ac.nz/ml/weka/>.